VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Klasifikace dokumentů s využitím GPU-PSO

The GPU-PSO Based Document Classification

2012 Bc. Tomáš Ježowicz

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Tomáš Ježowicz**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Klasifikace dokumentů s využitím GPU-PSO
GPU-PSO Based Document Classification

Zásady pro vypracování:

1.Student zpracuje přehled současného stavu využití GPU pro klasifikaci dokumentů pomocí PSO.
2. Na základě zpracovaného přehledu navrhne vlastní metodu.
3. Provede implementaci navržené metody.
4. Zvolí tesovací kolekce dat a provede experimenty.
5. Experimenty vyhodnotí. Zhodnotí přínos GPU.
1.

Seznam doporučené odborné literatury:

1. Ziqiang Wang; Qingzhou Zhang; Dexian Zhang; , "A PSO-Based Web Document Classification Algorithm," Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on , vol.3, no., pp.659-664, July 30 2007-Aug. 1 2007
2. David Kirk, Wen-mei Hwu, Programming Massively Parallel Processors, MORGAN KAUFMANN, 2010.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **prof. RNDr. Václav Snášel, CSc.**

Datum zadání: 19.11.2010

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 2.5.2012 .....

Podpis

Poděkování

Rád bych poděkoval vedoucímu diplomové práce prof. RNDr. Václavu Snášelovi CSc. za odbornou pomoc a konzultaci při vytváření této práce.

Abstrakt

Tato diplomová práce se zabývá problémem klasifikace vektorů, se zaměřením na kategorizaci textů pomocí relativně nového stochastického evolučního algoritmu Particle Swarm Optimization (PSO). Tento algoritmus byl použit na problému klasifikace textů v článku [4], kde dosáhl nejlepších výsledků v porovnání s jinými uvedenými algoritmy na klasifikaci. Algoritmus a celý proces je však výpočetně velice náročný. Primárním cílem této práce bylo navrhnout a implementovat vylepšení za účelem zvýšení rychlosti výpočtu. Zrychlení doby výpočtu bylo možné díky využití paralelních výpočtů na GPU. Implementace v prostředí CUDA využívá vysokého výpočetního výkonu grafických karet ve zpracování paralelních úloh. Bylo dosaženo několikanásobného urychlení celkové doby výpočtu vykonáním kritických částí na GPU. Dále je zde uveden současný stav klasifikace pomocí PSO a na GPU. V této práci jsou také zhodnoceny provedené experimenty a výsledky.

Klíčová slova

klasifikace, kategorizace, Reuters, 20 Newsgroup, Iris, PSO, nVidia, GPU, CUDA

Abstract

This thesis deals with the vector classification problem, focusing on text categorization using the relatively new stochastic evolutionary algorithm, Particle Swarm Optimization (PSO). This algorithm was used on the text classification problem in [4], where he achieves the best results in comparison with other algorithms mentioned there. The algorithm and the whole process is very computationally demanding. The primary goal of this thesis was to design and implement improvement in order to speed up whole process. The calculation speed up was possible because of parallel computing on GPUs. Implementation in CUDA uses the high computing performance of graphic cards in parallel processing tasks. Multiple acceleration of the process was achieved by executing critical parts on the GPU. The current state of the PSO classification and classification on the GPU is given. Performed experiments and results are evaluated.

Key words

classification, categorization, Reuters, 20 Newsgroup, Iris PSO, nVidia, GPU, CUDA

Seznam použitých symbolů a zkratek

CUDA	Computed Unified Device Architecture
DTA	Decision-tree algorithm
GPU	Graphic processing unit
HW	Hardware
Kernel	Funkce na GPU
KNN	K-nearest neighbor
NBC	Naive Bayes classifier
PSO	Particle Swarm Optimalization
SDK	Software Development Kit
SVM	Support Vector Machine
Threshold	Mezní hodnota
WSD	Word sense disambiguation

Seznam tabulek

Seznam obrázků

Obsah

Úvod

Klasifikace (kategorizace) dokumentů umožňuje automaticky třídit dokumenty do předem určených kategorií. Tato úloha má velmi mnoho aplikací, jako například automatické zařazování vědeckých článků, filtrování spamů, třídění dokumentů. Tento přístup osvobozuje od nutnosti manuálního roztřiďování. Particle Swarm Optimization (PSO) je relativně nový stochastický evoluční algoritmus, který umožňuje efektivní optimalizaci různých tříd problémů. Využívá jedince, kteří reprezentují řešení problému. Tito jedinci mezi sebou spolupracují a vyhledávají tak v prostoru nejlepší řešení. Klasifikaci dokumentů můžeme chápat jako optimalizační úlohu, tedy můžeme využít PSO. Samotná optimalizace je však velmi výpočetně náročná, ale některé úlohy v klasifikování pomocí PSO se dají provádět paralelně. V současné době existuje možnost využít vysoký výkon ve zpracování paralelních úloh, kterým disponují moderní grafické karty (GPU). S využitím GPU a prostředí CUDA můžeme některé výpočty řešit efektivněji, než na běžném CPU. Cílem práce je tedy provést implementaci a experimenty nového přístupu ke klasifikaci dokumentů a pokusit se urychlit tento výpočet s pomocí GPU.

Vývoj výpočetní techniky se v dnešní době se stále více zaměřuje na větší počet jader. Grafické karty už několik let řeší specifické úlohy rychleji než klasické procesory. Dnes existují dva hlavní výrobci grafických karet, kteří mají odlišné varianty přístupu k programování na svých kartách. Prostředí CUDA (Computed Unified Device Architecture), které vyvíjí společnost nVidia, je nástroj pro práci s grafickými kartami vyrobené společností nVidia. Používá rozšířený jazyk C/C++ a je vyvíjeno pro současné GPU nVidia a pro superpočítače.

1 Předzpracování textů

1.1 Redukce dimenze

Předzpracováním dokumentů se odstraní nevýznamná slova. Některá slova se také dají zkrátit na základní tvar. Hlavní výhodou předzpracování je zmenšení počtu slov a tedy zredukování dimenze prohledávaného prostoru. Nevýhodou je samozřejmě ztráta části informace.

1.2 Odstranění stop slov

V prostém textu i v přirozeném jazyce se často vyskytují slova, která nenesou žádnou důležitou informaci. Jedná se převážně o předložky a spojky. Taková slova se při předzpracování textů označí jako stop slova a typicky se zcela ignorují. V angličtině se do stop slov zařazují například slova the, is, at, atd. V češtině jsou to slova jako a, na, ale, oni, my. Je třeba také zmínit, že pro angličtinu existují volně stažitelné seznamy takovýchto slov. Do tohoto seznamu se mohou přidat i další slova, která jsou typická pro konkrétní dokumenty. Pokud se konkrétní slovo v kolekci vyskytuje v každém dokumentu, pro další zpracování také určitě nemá žádný význam.

V této práci je použít seznam slov z [1]. U standardní testovací kolekce „Reuters“ bylo také přidáno slovo reuters, protože se vyskytuje ve většině článků. Seznam [1] má tyto slova:

```
a, about, above, after, again, against, all, am, an, and, any, are,
aren't, as, at, be, because, been, before, being, below, between,
both, but, by, can't, cannot, could, couldn't, did, didn't, do,
does, doesn't, doing, don't, down, during, each, few, for, from,
further, had, hadn't, has, hasn't, have, haven't, having, he, he'd,
he'll, he's, her, here, here's, hers, herself, him, himself, his,
how, how's, i, i'd, i'll, i'm, i've, if, in, into, is, isn't, it,
it's, its, itself, let's, me, more, most, mustn't, my, myself, no,
nor, not, of, off, on, once, only, or, other, ought, our, ours, our-
selves, out, over, own, same, shan't, she, she'd, she'll, she's,
should, shouldn't, so, some, such, than, that, that's, the, their,
theirs, them, themselves, then, there, there's, these, they, they'd,
they'll, they're, they've, this, those, through, to, too, under, un-
til, up, very, was, wasn't, we, we'd, we'll, we're, we've, were,
weren't, what, what's, when, when's, where, where's, which, while,
who, who's, whom, why, why's, with, won't, would, wouldn't, you,
you'd, you'll, you're, you've, your, yours, yourself, yourselves,
reuters
```

1.3 Lemmatizace

Z gramatických důvodů se slova vyskytují v různých pádech. Pro automatizované zpracování textů je vhodné znormalizovat každé slovo převedením na jeho základní tvar. Tento proces se nazývá lemmatizace a můžeme k němu přistoupit dvěma základními způsoby. První přístup spočívá v použití obrovského slovníku, který by mapoval všechna slova na jejich základní tvary. Výhodou je, že ke každému slovu známe přesně jeho základní tvar (najdeme ho ve slovníku). Nevýhodou je samozřejmě paměťová náročnost a špatná udržovatelnost takového slovníku (pokud se objeví nové slovo). Druhý přístup je pomocí speciálního algoritmu, který každé slovo převede na základní tvar pomocí určitých pravidel. Nevýhodou je, že algoritmus není vždy nejpřesnější. Výhodou naopak je, že si dokáže poradit i s novými slovy. V této práci byl pro lemmatizaci použit algoritmus Porter Stemmer [2], který je volně dostupný v mnoha programovacích jazycích.

1.4 Reprezentace dokumentů pomocí vektorů

Každý dokument se tedy rozdělí na jednotlivá slova, kterých může být velmi mnoho. Jednotlivá slova se porovnají se seznamem zakázaných slov (stop slov) [1]. Pokud se v dokumentu objeví slovo, které je v tomto seznamu, je ignorováno. Dále se každé jednotlivé slovo převede podle algoritmu Porter Stemmer [2] na základní tvar a dále se uchovává jen počet jednotlivých termů (slov a složených slov) v daném dokumentu.

Poté se vytvoří matice $(Doc_j \times TF_{jk})$, kde Doc_j jsou dokumenty v kolekci a TF_{jk} jsou výskyty termů v jednotlivých dokumentech. Nyní potřebujeme dokumenty vyjádřit pomocí vektorů. K tomu se použije postup, který je popsán například v [4].

$$v_{ij} = t_{ij} \times \log \frac{F}{f_j}$$

Pro každý dokument i se postupně spočítá v_{ij} tak, že se vynásobí t_{ij} (počet výskytů termů v dokumentu) s logaritmem zlomku, ve kterém se F (počet různých termů v kolekci) podělí počtem výskytů termu j v celé kolekci.

1.5 Výběr nejdůležitějších slov

I přes odstranění stop slov a lemmatizaci mohou být v kolekci desítky tisíc termů a ne všechny klasifikátory si dokážou poradit s tak velkým prostorem [4]. Aplikováním postupu pro výběr nejdůležitějších slov, zmenšíme původní prohledávaný prostor na velikost, která nám lépe vyhovuje. Postupů pro výběr nejdůležitějších slov existuje sice více, ale aby se výsledky předešlých výzkumů daly lépe porovnat, zvolil jsem následující, který je v publikaci [4]. Významnost každého termu se vypočte násobením matice a vektoru $L_{jk} \times G_k$, kde L_{jk} jsou lokální váhy termů a G_k jsou globální váhy.

$$L_{jk} = \begin{cases} 1 + \log TF_{jk}: & TF_{jk} > 0 \\ 0: & TF_{jk} = 0 \end{cases}$$

$$G_k = \left(1 + \sum_{k=1}^{N} \frac{TF_{jk}}{F_k} \log \frac{\mathrm{TF_{jk}}}{\mathrm{F_k}}\right) / \log N$$

Kde index j prochází dokumenty a index k prochází všechny termy. TF_{jk} je počet výskytů každého slova v dokumentu. F_k je frekvence termu k v celé kolekci dokumentů. N je celkový počet dokumentů v kolekci. Vynásobením takto získané matice a vektoru získáme ohodnocení jednotlivých termů. Vybereme jen ty s nejvyšší hodnotou a zbytek ignorujeme. Tímto postupem se tak dá snížit dimenze prohledávaného prostoru z desítek tisíc například na hodnotu 300.

2 Klasifikace dokumentů

2.1 Definice klasifikace

Klasifikace dokumentů se dá chápat jako problém vyplnění tabulky hodnotami 0 a 1. Řádky tabulky tvoří předem definované kategorie a sloupce pak jednotlivé dokumenty. Pokud se v buňce tabulky objeví hodnota 1, pak to znamená, že příslušný dokument patří do kategorie a naopak, pokud se v buňce objeví 0, znamená to, že daný dokument nepatří do dané kategorie. Dokumentů ve sloupcích může být teoreticky nekonečné množství (např. s časem přibývající emaily). Klasifikace dokumentů je tedy problém, jak rozhodnout, do které kategorie dokument patří. Mějme tuto tabulku:

	$\boldsymbol{d_1}$	...	$\boldsymbol{d_j}$	...	$\boldsymbol{d_n}$
$\boldsymbol{c_1}$	a_{11}	...	a_{1j}	...	a_{1n}
⋮	⋮	...	⋮	...	⋮
$\boldsymbol{c_i}$	a_{i1}	...	a_{ij}	...	a_{in}
⋮	⋮	...	⋮	...	⋮
$\boldsymbol{c_m}$	a_{m1}	...	a_{mj}	...	a_{mn}

Tabulka 1 Přiřazování kategorií dokumentům [5]

Pak $D = \{d_1, \dots, d_n\}$ je množina dokumentů, které je třeba klasifikovat a $C = \{c_1, \dots, c_m\}$ jsou předem známé kategorie (někdy také třídy). Buňky a_{ij}, kde $i \in \{1 \dots m\}$ a $j \in \{1 \dots n\}$ nabývají hodnoty 0 a 1. Hodnota 1 znamená rozhodnutí, že dokument d_j patří do kategorie c_i. Hodnota 0 se interpretuje tak, že dokument d_j nepatří do kategorie c_i.

2.2 Trénovací a testovací množina

Celá oblast kategorizování dokumentů je založena na tom, že existuje předem daná sada dokumentů, která je správně roztříděna do předem známých kategorií. O každém dokumentu rozhodl nějaký expert, že patří, nebo nepatří do příslušné kategorie. Pokud dokument d_j patří do kategorie c_i, můžeme o něm říct, že je pozitivní příklad pro kategorii c_i. Pokud d_j nepatří do kategorie c_i, označuje se tento dokument jako negativní příklad kategorie c_i.

	Trénovací množina			**Testovací množina**		
	$\boldsymbol{d_1^{tr}}$	...	$\boldsymbol{d_s^{tr}}$	$\boldsymbol{d_{(s+1)}^{te}}$	...	$\boldsymbol{d_n^{te}}$
$\boldsymbol{c_1}$	a_{11}^{tr}	...	a_{1s}^{tr}	$a_{1(s+1)}^{te}$	...	a_{1n}^{te}
⋮	⋮	...	⋮	⋮	...	⋮
$\boldsymbol{c_i}$	a_{i1}^{tr}	...	a_{is}^{tr}	$a_{i(s+1)}^{te}$	...	a_{in}^{te}

⋮	⋮	…	⋮	⋮	…	⋮
c_m	a_{m1}^{tr}	…	a_{ms}^{tr}	$a_{m(s+1)}^{te}$	…	a_{mn}^{te}

Tabulka 2 Trénovací a testovací množiny dokumentů

V počáteční fázi se množina dokumentů ohodnocených expertem rozdělí na trénovací množinu $Tr = \{d_1^{tr} \dots d_s^{tr}\}$ a testovací množinu $Te = \{d_{(s+1)}^{te} \dots d_n^{te}\}$. Testovací množina se používá pro zjišťování efektivity klasifikátoru. Vyhodnocení efektivity klasifikátoru se provádí na základě srovnání vlastního vyhodnocení s hodnocením experta. Pro tvorbu a testování klasifikátoru musíme vědět, do jakých kategorií každý dokument patří. Dokážeme tedy jednoznačně určit, zda konkrétní dokument d patří nebo nepatří do kategorie c.

Množina Te by žádným způsobem neměla ovlivňovat tvorbu klasifikátoru. Jinak by se mohlo stát, že by výsledky experimentů byly nerealistické. Pokud je třeba vyladit nějaké parametry klasifikátoru, měla by se použít takzvaná validační testovací množina, která vznikne přesunutím vybraných dokumentů z trénovací množiny. Trénovací, validační a testovací množiny nemusí mít nutně stejnou velikost. Můžeme použít například rozdělení, kde je 60% dokumentů v trénovací, 20% ve validační a 20% v testovací množině.

2.3 K-fold cross validation

Jednou z alternativ k rozdělení kolekce na trénovací, validační a testovací je přístup K-fold cross validation [8]. Ten spočívá v tom, že datovou kolekci náhodně rozdělí do k-podmnožin. Podmnožiny jsou navzájem disjunktní a mají zhruba stejnou velikost. Každá podmnožina je pak použita právě jednou jako testovací. Zbylé podmnožiny se použijí k natrénování klasifikátoru. Tento postup se opakuje pro každou podmnožinu. Vznikne nám tedy k-výsledků, které se zprůměrují, a tím získáme celkové vyhodnocení efektivity klasifikátoru.

2.4 Definice threshold

Existuje mnoho různých možností, jak určit threshold pro jednotlivé kategorie. Jednou z možností je určit si jednu fixní hodnotu pro všechny kategorie. Tento přístup byl zmíněn v [10]. To ale podle autora může vést k tomu, že se všechny testovací dokumenty zařadí do kategorie c_A, zatímco žádný do kategorie c_B. V [1] se autor snaží najít nejlepší threshold pro každou kategorii zvlášť. Každý threshold je optimalizován pomocí validační množiny a je vybrán ten, se kterým se dosáhne nejlepší hodnoty účelové funkce.

2.5 Vyhodnocení klasifikátoru

Efektivita klasifikátoru se vyhodnocuje pomocí testovací množiny. Existuje mnoho různých kritérií, podle kterých lze hodnotit klasifikátory. Nejjednodušší je spočítat počet správně a nesprávně vyhodnocených dokumentů. Jednoduchou úpravou pak lze spočítat procentuální úspěšnost.

2.5.1 Precision a Recall

Sofistikovanějším způsobem měření efektivity klasifikátoru je evaluační funkce F_1. Předtím si nadefinujme precision a recall [5]. Precision (Pr) je definována jako pravděpodobnost, že náhodně vybraný dokument je správně klasifikován do správné kategorie [8]. Recall (Re) je pravděpodobnost zařazení náhodně vybraného dokumentu do kategorie, pokud by tam dokument měl být zařazen [8].

$$Pr_i = \frac{TP_i}{TP_i + FP_i}$$

$$Re_i = \frac{TP_i}{TP_i + FN_i}$$

TP (true positive), je definována jako suma všech dobře klasifikovaných dokumentů. FP (false positive), je počet všech dokumentů, které byly označeny klasifikátorem, že patří do kategorie, ale ve skutečnosti nepatří. FN (false negative) je počet dokumentů, které klasifikátor chybně označil, že do kategorie nepatří. Nakonec TN (true negative) je počet správně klasifikovaných dokumentů, které do kategorie nepatří.

	Dokument **patří** do kategorie c	Dokument **nepatří** do kategorie c
Klasifikátor říká, že dokument **patří** do kategorie c.	TP_c	FP_c
Klasifikátor říká, že dokument **nepatří** do kategorie c.	FN_c	TN_c

Tabulka 3 Klasifikace kategorie c klasifikátorem a expertem.

Nyní můžeme definovat funkci F_1[9].

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re}$$

2.5.2 Micro a Macro average

Pokud chceme vyhodnocovat efektivitu klasifikátoru v klasifikování více kategorií ve stejné kolekci, musíme nějak zprůměrovat precision a recall [10]. Mějme tedy dokumenty v kolekci a mějme je vyhodnocené pomocí F_1 (zda patří nebo nepatří do jednotlivých kategorií). Pak macroaverage je definován jako aritmetický průměr ohodnocení všech kategorií. To vede k zvýhodnění kategorií, které jsou méně početné [4]. Naproti tomu microaverage zvýhodňuje kategorie s větším počtem dokumentů. Výpočet tedy probíhá proporcionálně vzhledem k počtu pozitivních příkladů (dokumentů) jednotlivých kategorií [10].

	Microaverage	**Macroaverage**
Precision (Pr)	$Pr = \frac{\sum_{i=1}^{\|C\|} TP_i}{\sum_{i=1}^{\|C\|} TP_i + FP_i}$	$Pr = \frac{\sum_{i=1}^{\|C\|} Pr_i}{\|C\|} = \frac{\sum_{i=1}^{\|C\|} \frac{TP_i}{TP_i + FP_i}}{\|C\|}$
Recall (Re)	$Re = \frac{\sum_{i=1}^{\|C\|} TP_i}{\sum_{i=1}^{\|C\|} TP_i + FN_i}$	$Re = \frac{\sum_{i=1}^{\|C\|} Re_i}{\|C\|} = \frac{\sum_{i=1}^{\|C\|} \frac{TP_i}{TP_i + FN_i}}{\|C\|}$

Tabulka 4 Výpočet Precision a Recall (Microaverage a Macroaverage) [10]

2.6 Příklady využití klasifikace

Klasifikace textů se dá použít kdekoliv, kde je potřeba nějakým způsobem zorganizovat dokumenty. Automatizované organizování dokumentů je možné použít například v redakcích novin, při archivování článků, na patentovém úřadě, pro třídění patentů. Organizování dokumentů do předem známých kategorií lze obecně použít v jakémkoliv uložišti dokumentů. Například v už zmiňovaných novinách si lze představit potřebu automatického zařazování reklam do kategorií.

Ještě před několika lety v internetu převládaly vyhledávače, které byly založené na katalozích webových stránek. Nejznámějším příkladem takového vyhledavače je asi vyhledavač „Yahoo!“. Nejznámějšími českými katalogy jsou Seznam.cz, který má například katalog firem, dále Centrum.cz, nebo Atlas.cz. Pokud jsou webové dokumenty katalogizovány tímto způsobem, tak webový vyhledávač může uživateli nabídnout hierarchické procházení jednotlivými kategoriemi. V případě potřeby kategorizovat tak velký počet dokumentů, je téměř nepředstavitelné to dělat manuálně.

Dalších příkladů využití klasifikace je celá řada. Za zmínku stojí rozpoznávání nevyžádané pošty (spamů), filtrování textů (na relevantní, nerelevantní), nebo rozpoznávání významu slov (problém WSD)

3 Particle Swarm Optimization

3.1 Úvod do evolučních optimalizačních algoritmů

Evoluční optimalizační algoritmy jsou jen několik desítek let staré a neustále se vyvíjející metody pro řešení libovolného optimalizačního problému. Jako na optimalizační problém můžeme pohlížet na velkou skupinu reálných i čistě teoretických problémů. V literatuře [6] se dělí optimalizační algoritmy do čtyř základních skupin: enumerativní, deterministické stochastické a smíšené.

Enumerativní algoritmus vypočte každou možnou kombinaci vstupů a tím získá nejlepší řešení. Je jasné, že enumerativní metody zdaleka nejsou vhodné pro všechny problémy. U složitějších funkcí, které mají větší počet parametrů (dimenzí) se tento přístup nedá použít vůbec. Deterministické algoritmy využívají klasickou matematiku. Stejně jako enumerativní metody jsou deterministické metody z velké části omezeny. Většinou musíme znát například gradient, prohledávaný prostor musí být dostatečně malý a souvislý, problém má jen jedno nejlepší řešení, apod. Stochastické algoritmy jsou pomalé a založené na náhodě. Ale zároveň mohou sloužit například pro rychlý odhad toho, jak prohledávaný prostor vypadá. Nejznámějším stochastickým algoritmem je asi Random walk, který je založen na generování čistě náhodných řešení, které pak vyhodnocuje.

Poslední skupinou jsou smíšené metody, které kombinují deterministické a stochastické metody a dosahují překvapivě dobrých výsledků [6]. Ty se používají pro optimalizaci mnohaparametrových funkcí, které by klasickými dnes známými algoritmy nebyly řešitelné. Využívají náhodu, ale nejsou závislé na počátečních nastaveních. Mají společné to, že jsou nezávislé na optimalizované funkci. Jediné co je pro ně směrodatné je hodnota účelová funkce, kterou se snaží minimalizovat respektive maximalizovat. Jsou do jisté míry odolné proti uváznutí v lokálním extrému a snaží se najít globální extrém. Do této kategorie můžeme zařadit jistě dobře známé algoritmy, jako jsou Genetické algoritmy, algoritmus Ant Colony Optimalization, Diferencionální evoluci a v neposlední řadě Particle swarm optimalization.

3.2 Popis PSO

Tento algoritmus byl vyvinut pány Eberhartem a Kennedym a byl poprvé uveřejněn v roce 1995 [4]. PSO je populačně založená technika, která se inspiruje pohybem a chováním zvířat v hejnu (například ptáků). Každý jedinec (nebo také částice), představuje jednu pozici v optimalizovaném prostoru. Tedy jedno nastavení všech parametrů účelové funkce. Účelová funkce je matematické vyjádření problému, který máme optimalizovat. Při optimalizaci hledáme její minimum (resp. maximum). Po dosazení všech parametrů jedince do argumentů účelové funkce, získáme jeho vhodnost (ohodnocení).

Na začátku algoritmu se vytvoří náhodná populace tvořená jedinci. Pozice i-tého jedince v prostoru je reprezentovaná vektorem $x_i = \{xp_1^i, \dots, xp_j^i, \dots, xp_d^i\}$, kde d je dimenze účelové funkce a je dána optimalizovaným problémem, $j \in \{1 \dots d\}$ je index jednotlivých argumentů účelové funkce, xp_j^i odpovídá konkrétní hodnotě j-tého argumentu v i-tém jedinci. V každém kroku jedinci vyhodnotí své řešení,

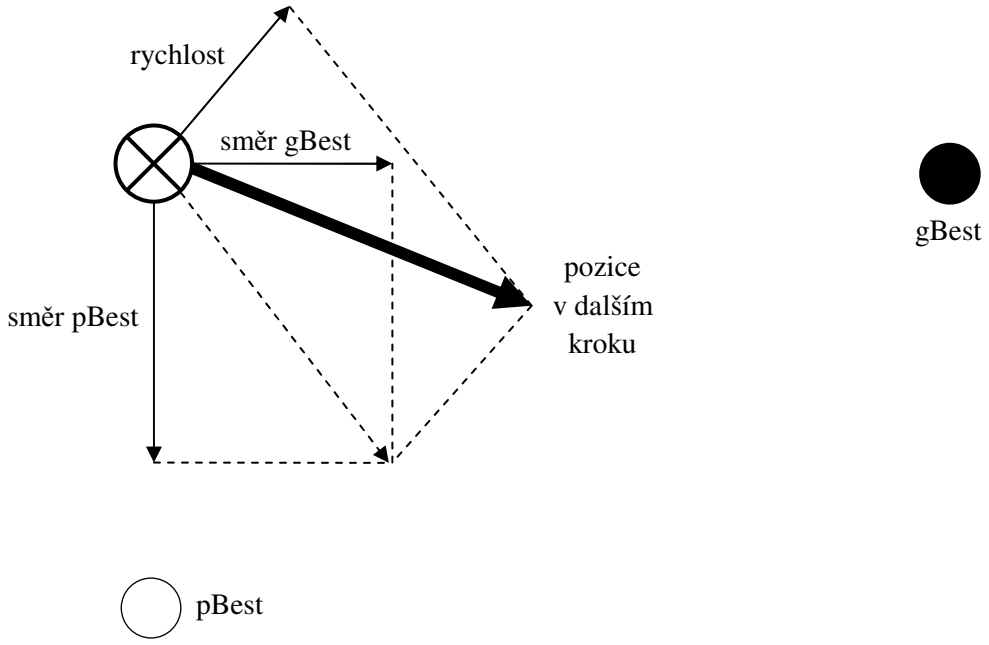
dosazením svých parametrů do účelové funkce. Jedinec si pamatuje nejen svou aktuální pozici, ale i své dosavadní nejlepší ohodnocení účelové funkce. Všichni jedinci znají i pozici nejlepšího jedince v populaci. Jedinci v dalším kroku (v další generaci) směřují buď ke své dosavadní nejlepší pozici, nebo k aktuálnímu globálnímu nejlepšímu řešení. Rychlost s jakou se k nové pozici každý jedinec pohybuje, se dá vyjádřit vektorem $v_i = \{vp_1^i, \dots, vp_j^i, \dots, vp_d^i\}$. Rychlost je na začátku stejně jako pozice vygenerována čistě náhodně. Aktualizaci pozice a rychlosti jedinců v dalším kroku zachycuje rovnice:

$$v_i(t+1) = w.v_i(t) + c_1.r_1.(pBest_i - x_i(t)) + c_2.r_2.(gBest - x_i(t))$$

Tato rovnice také obsahuje dvě náhodně vygenerovaná čísla r_1 a r_2, které mají rozsah od 0 do 1. Rychlost jedince v dalším kroku $v_i(t+1)$ se rovná součtu vektoru rychlosti v aktuálním kroku $v_i(t)$ s vektorem ukazujícím na svou dosavadní nejlepší dosaženou pozici $pBest_i$ a s vektorem, který ukazuje na globální nejlepší pozici $gBest$. K pozici jedince se přičte rychlost podle rovnice:

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1)$$

Možný pohyb jednoho jedince je ilustrován na obrázku 1, který je zobrazen jako kolečko s křížkem. Plné kolečko gBest představuje pozici aktuální nejlepšího globálního řešení. Bílé kolečko pBest je dosavadní nejlepší pozice jedince. Tlustá šipka je součet tří vektorů. Vektoru rychlosti jedince, vektoru směru k gBest a směru k pBest. Tím určuje pozici (nastavení parametrů) v dalším kroku.



Obrázek 1Ilustrace pohybu jednoho jedince v PSO.

3.3 Parametry a jejich nastavení

PSO má větší počet parametrů, které je třeba nastavit. Jejich nastavení se provádí experimentálně. Pro různé problémy může (a často se to stává) změna nastavení parametrů způsobit zlepšení ve smyslu nejlepšího nalezeného řešení. Nastavení parametrů tedy výrazně ovlivňuje chod algoritmu. Toto je problém stejně jako i u ostatních evolučních algoritmů [6].

Algoritmus skončí po předem daném počtu generací. Učící faktory c_1 a c_2 mají obvykle hodnotu 2. Mohou do jisté míry ovlivnit tendenci k gBest a pBest. Parametr setrvačnosti w se obvykle lineárně snižuje podle rovnice 4.3, kde w_{max} je maximální hodnota w, w_{min} je minimální hodnota w. Dle doporučení [6] se obvykle tyto hodnoty nastavují na 0,9 a 0,4. Malá hodnota w podporuje hledání lokálních extrémů a velká hodnota hledání globálních extrémů [6].

$$w = w_{max} - \frac{(w_{max} - w_{min})}{iter_{max}} \cdot iter$$

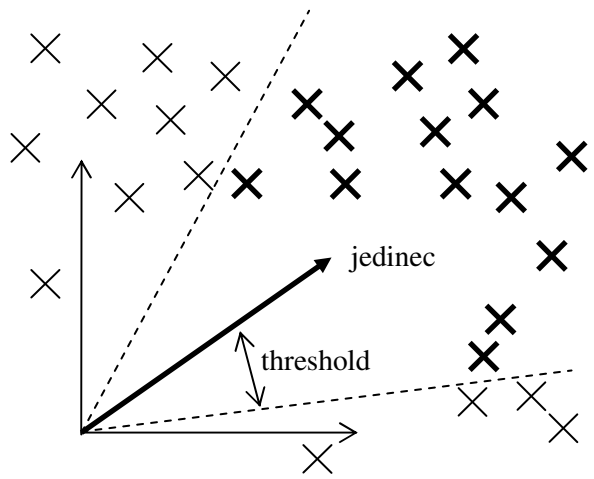
Dalšími parametry mohou být rozsahy jednotlivých dimenzí. Pro každou dimenzi mohou být definovány jiné množiny (oblasti) přípustných hodnot. Velikost populace je obecně doporučeno volit deset krát větší než je počet dimenzí prohledávaného prostoru. Posledním parametrem algoritmu je maximální rychlost částic v_{max}. V [6] se doporučuje, aby tento parametr měl hodnotu 1/20 rozsahu.

Existuje samozřejmě více variant tohoto algoritmu, které mohou přinést zlepšení na určité specifické úlohy.

3.4 Použití PSO na problém klasifikace dokumentů

Výstupem algoritmu PSO, stejně jako celé třídy evolučních algoritmů (Simulované žíhání, genetické algoritmy, SOMA, atd.) je obecně vektor s nejlepší hodnotou účelové funkce. Stačí tedy vhodně zvolit účelovou funkci a algoritmus pak vyhledává nejlepší řešení. Účelová funkce pro problém klasifikace textů je definována výše funkcí F_1. Pro výpočet funkce musíme znát precision a recall. Ty vypočteme na základě počtu správně a nesprávně klasifikovaných dokumentů. Samotné rozhodnutí, zda dokument patří nebo nepatří do kategorie c, je záležitostí porovnání dvou vektorů.

Mějme n-jedinců v PSO. Každý jedinec hledá takový vektor, který co nejlépe charakterizuje celou kategorii c. Při tomto procesu se v každé iteraci porovnává pozice jedince (vektor) s celou trénovací množinou dokumentů (vektorů). Porovnáním dvou vektorů dostaneme číselné vyjádření podobnosti dvou vektorů a to porovnáme s číselnou konstantou (threshold). Pokud je dokument dostatečně podobný jedinci (podobnost > threshold), tak je tento dokument klasifikován, že patří do kategorie c. Rozpoznávání je názorně vidět na obrázku 2. Křížkem jsou znázorněny dokumenty. Tlustěji jsou vyznačeny dokumenty, které ještě patří do kategorie. Hraniční hodnota threshold je čárkovaně. Pozice jedince je znázorněna tlustou šipkou.

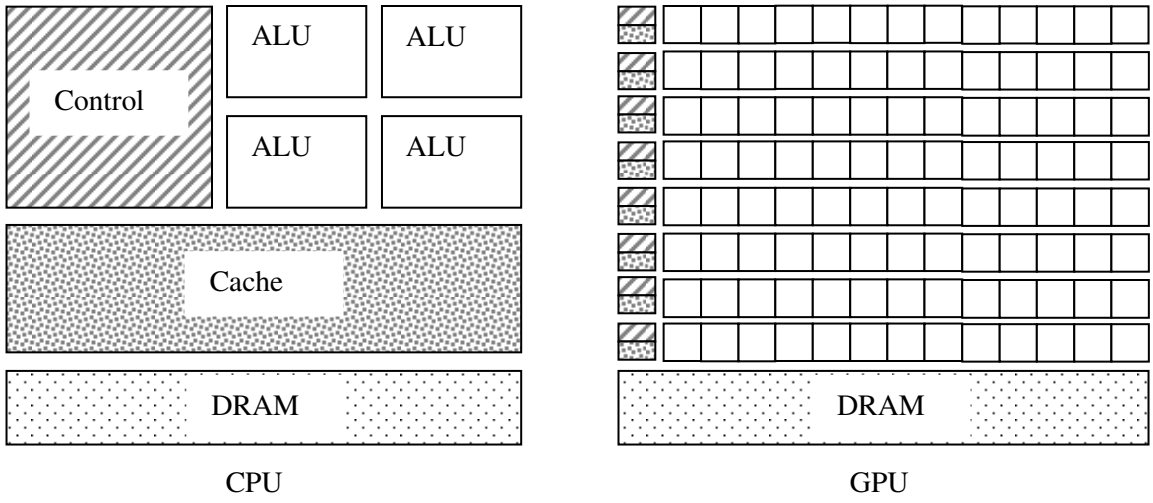


Obrázek 2 Rozpoznávání dokumentů patřících do kategorie.

4 Výpočty na GPU

4.1.1 Základní odlišnosti GPU a CPU

GPU je primárně určeno pro vysoce paralelní výpočty, jako je tvorba reálného obrazu na základě počítačového modelu (randering). Proto mají GPU více tranzistorů zaměřených více na zpracování dat (data processing), než na udržování vyrovnávací paměti (data caching) a řízení datových toků (flow control) [3]. Schematicky je to znázorněno na obrázku 3. Prostředí umožňuje využít výhod takzvaného heterogenního programování, což jednoduše znamená, že část kódu se provede na GPU a část na CPU.



Obrázek 3 Základní odlišnost CPU a GPU [3].

4.1.2 Kopírování paměti

Grafická karta má více druhů pamětí. Nejdůležitější jsou globální paměť, sdílená paměť a registry. Mezi nejrychlejší paměti patří sdílená paměť. Rychlejší jsou jen registry. Sdílená paměť umožňuje spolupráci v rámci jednoho „bloku“ a má velikost 16KB. Funkce cudaMalloc alokuje paměť na GPU.

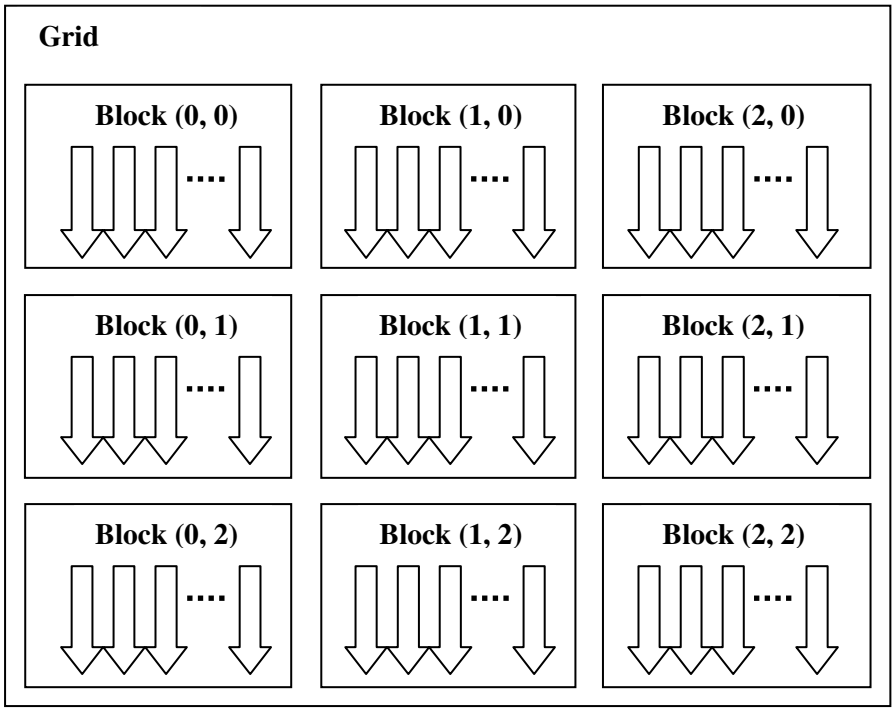
```
cudaMalloc((void**)&A_d, size);
```

Data ke zpracování se do GPU a zpět posílají přes globální paměť pomocí funkce cudaMemcpy:

```
cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
```

4.1.3 Kernel a vlákna

Kernel je funkce, která je prováděna na grafické kartě (device). Všechny vlákna (threads) zpracovávají tuto funkci paralelně. Respektive se chovají tak, jako by běžely paralelně. Fyzicky se grafické kartě provádí paralelně jen určitý počet bloků (blocks).



Obrázek 4 Uspořádání bloků v grid.

Každé volání __global__ funkce musí být předem nakonfigurováno. Tato konfigurace určí velikost grid, bloků a počet vláken v jednotlivých blocích. Konfigurace se při volání funkce vkládá mezi její název a mezi parametry této funkce mezi závorky <<< >>>. Příklad konfigurace a volání kernelu:

```
dim3 dimBlock(M, N);
kernel<<<1, dimBlock>>>(A, B, C);
```

Každé vlákno má přiděleno svoje unikátní identifikační číslo, které je přístupné přes zabudovanou proměnnou threadIdx. Podobně každý blok má své číslo pod proměnnou blockIdx a dimenze všech použitých bloků je ukryta pod proměnnou gridDim. Kernel vlastně spouští grid, ve kterém jsou bloky, které obsahují samotné vlákna. Velikost grid závisí na uživatelské konfiguraci kernelu, kterou je nutno provést před každým jeho voláním. Příklad kernelu pro součet dvou vektorů:

```
__global__ void vecAdd(float* A, float* B, float* C)
{
      int i = threadIdx.x;
      C[i] = A[i] + B[i];
}
```

4.1.4 Streamy

Jednou z možností, jak vylepšit rychlost výpočtu na GPU je použít streamy. Příkazům jako kopírování paměti, nebo vykonávání výpočtu na GPU lze přiřadit stream. Tím je zaručeno, že se jednotlivé části provedou v pořadí za sebou. Síla streamů se ale projeví, až v okamžiku kdy je jich použito více najednou [12]. Pro použití streamů musíme alokovat speciální typ paměti (Page-Locked). Příklad alokace paměti na CPU straně:

```
cudaHostAlloc((void**)&h_a, SIZE * sizeof(int), cudaHostAllocDefault );
```

Pro kopírování paměti se používá speciální asynchronní funkce cudaMemcpyAsync():

```
cudaMemcpyAsync(d_b, h_b,N*sizeof(int),
                cudaMemcpyHostToDevice,
                stream);
```

Samotný stream se vytvoří jednoduše takto:

```
cudaStream_t stream;
cudaStreamCreate(&stream );
```

Výpočet na GPU se pomocí streamu provede přidáním parametru do konfigurace kernelu:

```
kernel<<<N/256,256,0,stream>>>(d_a, d_b, d_c);
```

Protože streamy pracují asynchronně, je třeba je synchronizovat funkcí cudaStreamSynchronize(). Tato funkce zaručuje, že se všechny operace provedou a od této chvíle je možné s daty bezpečně pracovat:

```
cudaStreamSynchronize(stream);
```

Nesmíme zapomenout na odstranění streamu:

```
cudaStreamDestroy(stream);
```

4.1.5 Využití více GPU

V předchozích verzích CUDA bylo nutné pro použití více GPU vytvářet CPU vlákna. V nejnovější verzi je použití více grafických karet velice jednoduché. Stačí vytvořit v každé kartě stream [15]. Jediné na co je třeba si dávat pozor je přepínání jednotlivých karet před každým příkazem funkcí cudaSetDevice(„Číslo GPU“)

```
cudaSetDevice( 0 ); cudaStreamCreate( &streamA );
cudaSetDevice( 1 ); cudaStreamCreate( &streamB );
cudaSetDevice( 0 ); kernel<<<..., streamA>>>(...);
```

5 Současný stav klasifikace PSO a GPU

5.1 Klasifikace pomocí PSO

Jak bylo zmíněno v úvodu, tato diplomová práce vznikla hlavně na základě publikace [4]. Zde byl algoritmus PSO použit na problém klasifikace textu a otestován na kolekcích Reuters a TREC corpora. Klasifikace pomocí PSO byla srovnána se třemi dalšími algoritmy pro klasifikaci (DTA, NBC, KNN). Práce bohužel neobsahuje informace o časové náročnosti experimentů. Jak je vidět v tabulce 5, tak PSO zde ve srovnání s výše uvedenými algoritmy dosahuje na kolekci Reuters nejlepších výsledků.

Metoda	**Macro-F1**	**Micro-F1**
DTA	0.7847	0.8541
NBC	0.6575	0.7906
KNN	0.6738	0.7815
PSO	0.8486	0.9108

Tabulka 5 Výsledky F1 skóre na kolekci Reuters dosáhnuté v [4].

5.2 Výběr feature pomocí PSO

Výběr nejdůležitějších slov nebo také feature selection znamená vybírání určité podmnožiny slov ze slov, které jsou v testované kolekci. Publikace [17] se zaměřuje právě na tento problém. Srovnávají se zde klasické přístupy s použitím algoritmu PSO. Byl proveden test na kolekci Arabic dataset a přístup PSO na této kolekci dosáhl nejlepších výsledků. Zde je vidět síla a univerzálnost algoritmu PSO. Algoritmus totiž nebyl použit pro hledání vektorů, které nejlépe charakterizují danou kategorii, ale pro výběr nejdůležitějších slov. V tabulce 6 lze vidět srovnání se standardními algoritmy pro výběr slov.

Algoritmus pro výběr nejdůležitějších slov	**Macro-F1**
CHI	85,5
DF	79,0
TFxIDF	92,1
PSO	93,9
W/O FS	65,0

Tabulka 6 Macro-F1, výběr nejdůležitějších slov pomocí PSO v kolekci Arabic.

5.3 Zrychlení PSO pomocí GPU

V publikaci [16] se autorům podařilo urychlit výpočet PSO pomocí GPU. Testování zde bylo provedeno na čtyřech standardních funkcích, kde každá měla své minimum v nule. Maximální zrychlení

bylo jedenáctinásobné (na testovaném HW) Tabulka 7 ukazuje maximální dosažené zrychlení při různých velikostech vstupů na konkrétní testovací funkci.

Funkce	Dimenze	Počet částic	Čas CPU (s)	Čas GPU (s)	Zrychlení
$\sum_{i=1}^{D} x_i^2$	50	400	118,4688	31,2999	3,7850
	100	400	237,6160	60,5560	3,9239
	150	400	356,7932	89,5998	3,9821
	200	400	483,0046	119,9542	4,0266
	50	400	47,9893	12,8183	3,7438
	50	1200	153,4973	33,0714	4,6414
	50	2000	250,3312	55,1357	4,5402
	50	2800	362,7370	72,7750	4,9843

Tabulka 7 Zrychlení algoritmu PSO na GPU v publikaci [16] na testovací funkci.

5.4 SVM na GPU

V diplomové práci [18] se autor zaměřil na zrychlení existující implementace algoritmu SVM na GPU. Zrychlení oproti implementaci algoritmu ve standardní knihovně LIBSVM se pohybuje od 89 až 263 násobku. Dosáhl zde také 1,24 až 2,35 násobného zrychlení algoritmu „K-nearest neighbors“ oproti předcházející implementaci na GPU. Tato diplomová práce se však zabývá klasifikací pomocí PSO. Srovnání zrychlení na GPU algoritmů SVM a K-nearest neighbors je zde spíše jen pro úplnost.

6 Návrh implementace

6.1 Předzpracování dat

Aby bylo možno pracovat s dokumenty jako s vektory, musejí se na vektory nějakým způsobem převést. Existuje sice několik předzpracovaných a převedených kolekcí, ale já jsem zvolil cestu vlastní implementace samotného předzpracování. Předzpracované kolekce by měly výhodu menší pracnosti. Naopak výhoda vlastní implementace je větší kontrola nad prováděnými kroky s větší volností pro experimenty. Vytvořil jsem proto program v jazyce Java, který tuto úlohu provede. Pro každou další kolekci se musí přidat nová implementace zpracování konkrétní kolekce, ale většina kódu se dá samozřejmě použít znovu. Pro kolekci Reuters program přečte vstupní soubor a na základě jeho konfigurace převede kolekci do formátů, které jsou schopny přečíst další programy. Konfigurační parametry pro kolekci Reuters stručně popisuje tabulka 8. Předzpracování textů se dá popsat v těchto několika krocích:

1. Odstranění krátkých slov
2. Odstranění čísel
3. Převod na malá písmena
4. Odstranění stop slov
5. Použití algoritmu Porter Stemmer
6. Rozdělení dat na trénovací/testovací/validační množiny
7. Určení číselné hodnoty jednotlivých termů
8. Provedení výběru atributů (features)
9. Normalizace vektorů
10. Uložení kolekce do výstupního formátu

Tabulka níže obsahuje popis konfiguračních parametrů:

Název parametru	**Typ**	**Popis**
FILES_TO_PARSE	List	Cesty k jednotlivým souborům kolekce.
OUTPUT_PATH	String	Cesta k výstupu programu.
WORDS_SEPARATORS	List	Seznam oddělovačů slov.
SKIP_WORDS_SHORTER_THAN	Integer	Odstraní kratší slova než parametr.
REMOVE_NUMBERS	Boolean	Mají se odstranit čísla?
TO_LOWERCASE	Boolean	Májí se slova převést na malá písmena?
STOPWORDS_LIST	List	Seznam stop slov.
USE_PORTER_STEMMER	Boolean	Má se aplikovat algoritmus Porter Stemmer?
ALLOWED_CATEGORIES	List	Seznam kategorií, které se mají použít.
SKIP_DOC_CONTAINS_ NOT_ALLOWED_CATEGORIES	Boolean	Mají se ignorovat dokumenty, které nejsou v seznamu použitých kategorií.

TRAIN_TEST_SPLIT	Výčet	Jaké má být rozdělení na trénovací, validační, případně testovací množiny?
USE_VALIDATION_SET	Boolean	Má se vygenerovat i validační množina?
WEIGHTING_METHOD	Výčet	Jakou metodou se mají převést slova na čísla?
USE_TITLES_ADVANTAGE	Boolean	Využít výhody znalosti nadpisů v dokumentech?
MAX_FEATURES_COUNT	Integer	Maximální počet features (dimenze).
SAVE_METHOD	Výčet	Metoda pro uložení dokumentů do výstupního formátu
NORMALIZE_VECTORS	Boolean	Normalizovat vektory před uložením?

Tabulka 8 Popis konfiguračního souboru pro kolekci Reuters.

Z výše uvedených je zajímavý parametr ALLOWED_CATEGORIES. Zařadil jsem ho do konfigurace, protože umožňuje zvolit jen určité kategorie. V některých článcích [4] se vybírají jen některé kategorie. S tím souvisí parametr SKIP_DOC_CONTAINS_NOT_ALLOWED_CATEGORIES, který umožňuje ignorovat dokumenty obsahující nedovolené kategorie. Mějme v seznamu dovolených kategorií kategorie Grain a Wheat. Pokud dokument d má kategorie Grain a Money, tak při aktivaci tohoto parametru nebude d zařazen do výstupních vektorů, i když obsahuje dovolenou kategorii Grain. Experimentálně jsem zjistil, že zapnutím tohoto parametru lze zvýšit kvalitu výsledků. Atribut TRAIN_TEST_SPLIT určuje rozdělení dokumentů na podmnožiny. Hodnoty „ModLewis“, „ModApte“ a „ModHayes“ rozdělí dokumenty podle standardního rozdělení kolekce Reuters. Implementoval jsem také další mnou navržené rozdělení „My_60_20_20“. To rozdělí dokumenty náhodně do tří kategorií. Zhruba Šedesát procent do trénovací, dvacet procent do validační a dvacet procent do testovací množiny. Parametr USE_TITLES_ADVANTAGE je další mnou navržený atribut, který při zapnutí využije znalosti nadpisu v dokumentech. Při převodu slov na vektory program navíc uměle přidá slova, která jsou v nadpisu (a která tam už tedy jsou jednou obsažena). Tento parametr ale nepřinesl očekávané zlepšení. Dalším parametrem je SAVE_METHOD. Protože výstupy programu slouží jako vstupy pro jiné programy a další experimenty, bylo implementováno několik metod ukládání výstupních vektorů: „EACH_CAT_ONE_FILE“ – uloží každou kategorii do samostatného souboru, „ONE_BIG_FILE“ uloží všechny kategorie do jednoho souboru, „ONE_BIG_FILE_3_FOLD“ uloží soubory do 3 částí (folds), „WEKA_ARFF_ONE_FILE“ uloží vektory do formátu pro program Weka, „LIB_SVM“ uloží vektory do formátu pro program LibSVM, „WEKA_ARFF_SEPARATE_FILES“ uloží vektory do souborů podle kategorií do formátu pro program Weka.

Předzpracování kolekce 20 Newsgroup je podobné s tím rozdílem, že v konfiguračním souboru je nutno předat cestu ke složkám, ve kterých se nacházejí trénovací, respektive k testovací dokumenty.

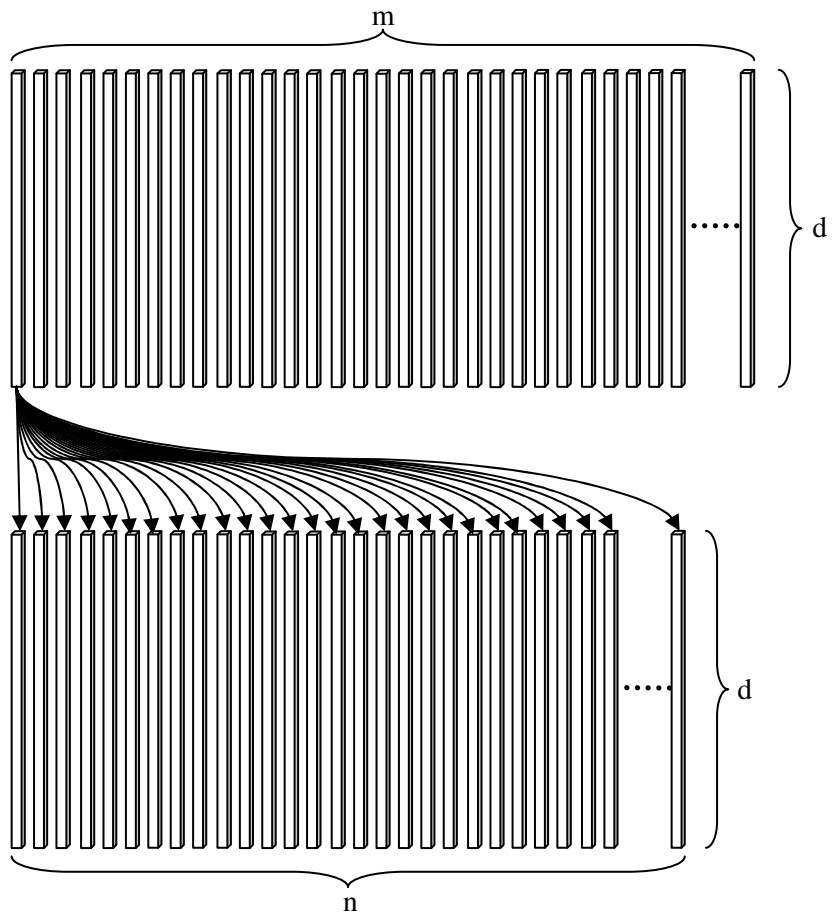
6.2 Popis problému urychlení klasifikace

Máme dokumenty převedeny do vektorů, víme, jak funguje PSO a jak ho nasadit na problém klasifikace. Nyní je třeba zjistit, která část výpočtu zabírá nejvíce výpočetního času, aby bylo možné navrhnout urychlení na GPU. Nejprve jsem tedy celý algoritmus naimplementoval na CPU. Poté časově porovnal dílčí úlohy.

Krok	Čas (ms)	Procentuální podíl (%)
PSO krok 1	1270	0,3414
Porovnání vektorů	365232	**98,1691**
Výpočet F1	5474	1,4713
PSO krok 2	68	0,0182

Tabulka 9 Srovnání dílčích časů výpočtu.

Časy dílčích úloh je vidět v tabulce 9. Tam je vidět, že nejvíce času zabírá výpočet, ve kterém se porovnává m vektorů dokumentů z kolekce s n vektory jedinců z PSO. Z toho vyplývá, že i kdybych se zaměřil na zrychlování algoritmu PSO, nebo dopočítání účelové funkce F1, čas výpočtu by se téměř nezměnil. Zatímco kdyby se podařilo jen dvakrát urychlit výpočet porovnání m vektorů s n, čas výpočtu by se zkrátil téměř o polovinu. Problém porovnávání je ilustrován v obr. 5. Protáhlý kvádr zde představuje vektor.



Obrázek 5 Ilustrace problému porovnání m vektorů s n vektory a s dimenzí d.

6.3 Implementace kernelu 1

Pro řešení problému porovnání vektorů tvořících kolekci s vektory jedinců algoritmu PSO jsem navrhnul dvě funkce na GPU. Každá z nich má svá specifika a výhody. Cílem obou funkcí je naplnit matici m x n, jejichž prvky jsou výsledky porovnání vždy dvou vektorů.

Mějme kolekci C tvořenou m vektory s dimenzí d a mějme kolekci n jedinců s dimenzí d, kterou nazveme P. První přístup spočívá v tom, že každé vlákno provede výpočet více porovnání dvou vektorů. Při konfiguraci kernel je potřeba alokovat m vláken. Každé vlákno pak vezme příslušný vektor z kolekce C a porovná s ním k vektorů z P. V obrázku 6, který zobrazuje tok dat v kernelu, je k rovno čtyřem. Vlákno musí projít d prvků pro k vektorů. Výstupem vlákna je k čísel (porovnání) ve výsledné matici.

Pseudokód konfigurace kernelu:

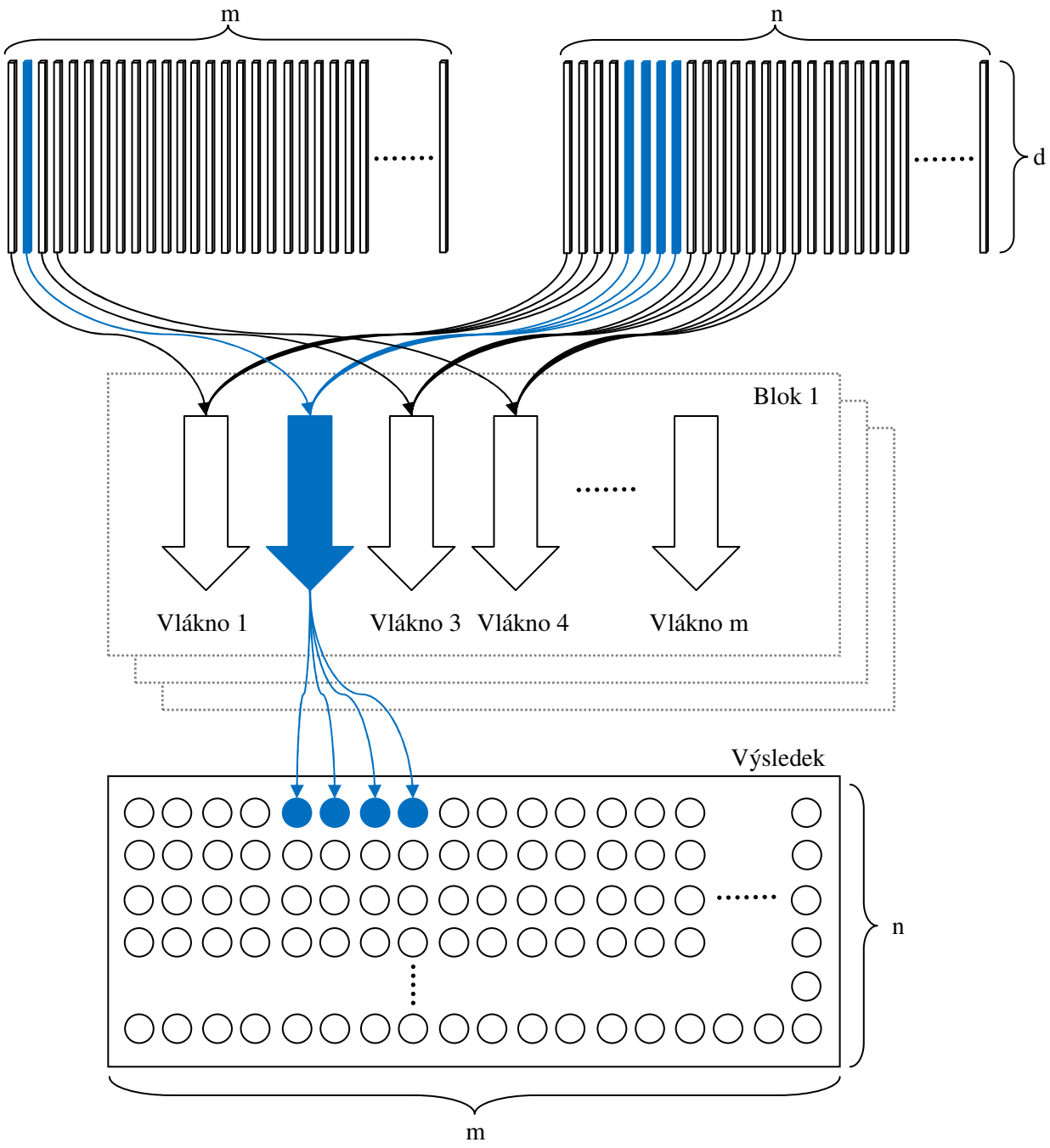
```
dim3  grid (((m + threads_size – 1) / threads_size), 1, 1);
dim3  threads (threads_size, 1, 1);
Kernel_1<<<grid, threads, 0>>>(C, P, Results, d, m, fromP, toP);
```

Pseudokód samotného kernelu:

```
int idx = blockDim.x * blockIdx.x + threadIdx.x;

while (idx < m)
{
      for(int k = fromP; k < toP; k++)
      {

            Results[k * m + idx] =  
```

$$\frac{\sum_{i=1}^{d} C_i \times P_i}{\sqrt{\sum_{i=1}^{d} (C_i)^2} \times \sqrt{\sum_{i=1}^{d} (P_i)^2}}$$

```
      }

      idx += blockDim.x * gridDim.x;
}
```



Obrázek 6 Vizualizace části výpočtu na GPU (kernel 1).

6.4 Implementace kernelu 2

Druhá metoda výpočtu, kterou jsem navrhnul, pohlíží na problém odlišně. Každé vlákno zde počítá právě jednu hodnotu prvku vektoru s dimenzí d. Vlákna pak mezi sebou (v rámci bloku vláken) dopočítají hodnotu jednoho prvku výsledné matice. Spolupráce se děje pomocí sdílené paměti. Chceme ze sdílené paměti, do které vlákna uložily své výpočty získat jen jednu výslednou hodnotu. Toho bylo docíleno tak, že každé druhé vlákno přistoupilo ke dvěma místům ve sdílené paměti, zbylá vlákna nedělala nic. Postup se opakuje, dokud poslední vlákno nevypočítá poslední (výslednou) hodnotu. Sdílená paměť musí být alokována tak, aby tvořila násobky dvou. Paměť, která je nadbytečná, se vynuluje. Pracujeme-li s dimenzi d, pak v jednom bloku vláken na GPU musí být nakonfigurováno alespoň d vláken. To je nevýhoda tohoto přístupu. Na testovaném HW je maximální nastavitelná velikost 1024 vláken pro jeden blok. Jak později ukážou experimenty, tento přístup je spíše vhodnější pro menší velikosti vstupu.

Pseudokód konfigurace a volání kernelu (threads_size musí být násobkem dvou a musí být větší než dimenze vektorů):

```
int  sh = threads_size * sizeof(double) * 3;
dim3 bl (threads_size);
dim3 gr (m, n);
Kernel_2<<< gr, bl, sh>>>( C, P, Results, m ,n , d);
```

Pseudokód první části kernelu 2, kde se vlákno počítá pomocí prvku z vektoru z C a prvku z P:

```
int idx = blockIdx.x * n + blockIdx.y;
extern __shared__ double numS[], den1S [], den2S[];
numS[threadIdx.x]=0; den1S[threadIdx.x]=0; den2S[threadIdx.x]=0;

if (threadIdx.x < d)
{
      double a = C[d * blockIdx.x + threadIdx.x];
      double b = P[n * threadIdx.x +blockIdx.y];

      numS [threadIdx.x] = a * b;
      den1S[threadIdx.x] = pow(a, 2);
      den2S[threadIdx.x] = pow(b, 2);
}
__syncthreads();
```

Pseudokód druhé části kernelu 2, ve které se postupně redukují vlákna, které provádí součet:

```
int i = blockDim.x / 2;

while (i != 0)
{
```

```
      if (threadIdx.x < i)
      {
            numS [threadIdx.x] += numS [threadIdx.x + i];
            den1S[threadIdx.x] += den1S[threadIdx.x + i];
            den2S[threadIdx.x] += den2S[threadIdx.x + i];
      }
      __syncthreads();
      i /= 2;
}
```

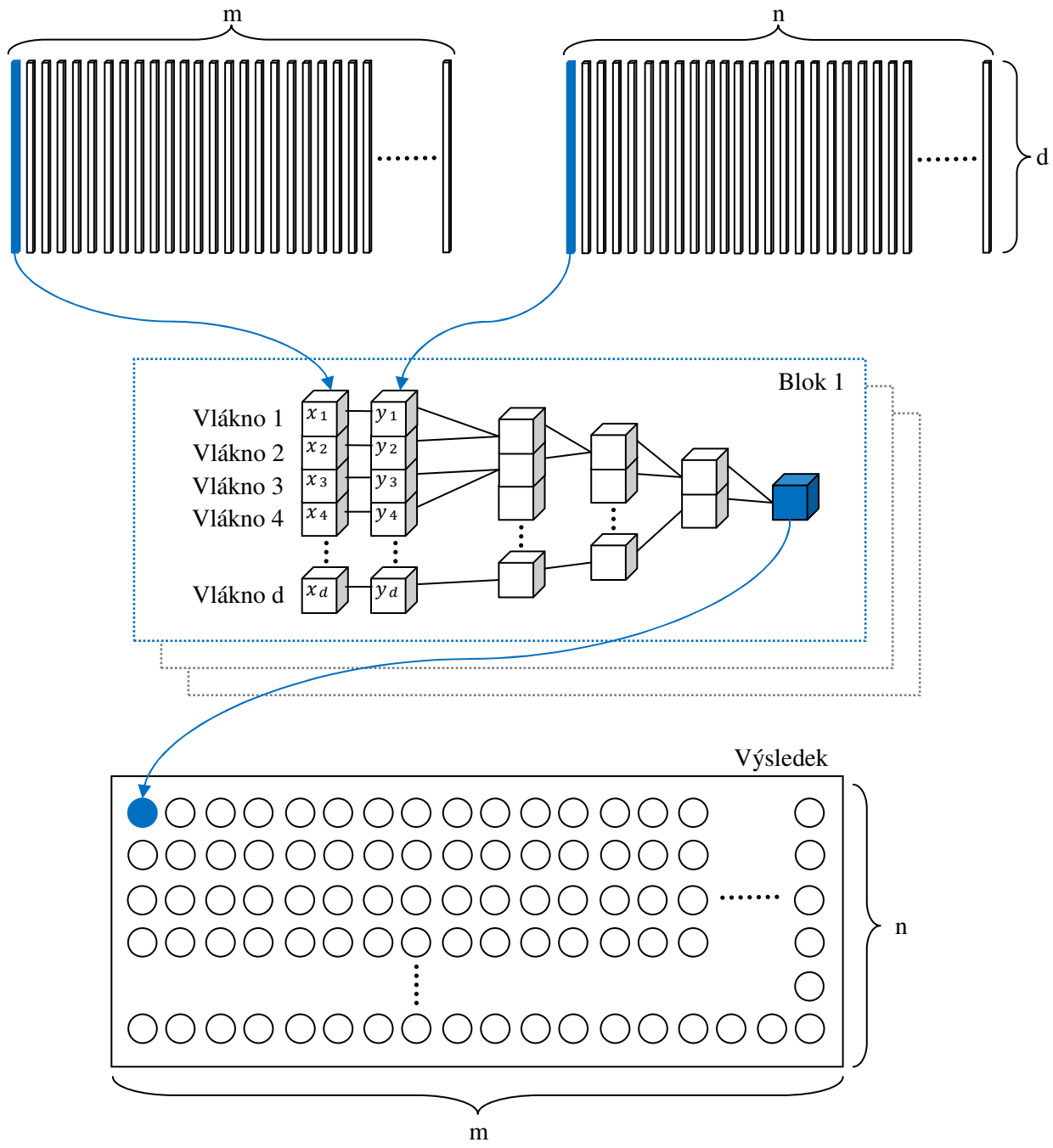
Na pozici 0 ve sdílené paměti je uložen konečný výsledek. Pomocí nultého vlákna se výsledek zapíše do globální paměti. Pseudokód poslední části kernelu 2:

```
if (threadIdx.x == 0)
{
```

$$\texttt{Results[idx]} = \frac{\text{numS}[0]}{\sqrt{\text{den1S}[0]} \times \sqrt{\text{den2S}[0]}}$$

```
}
```

Tok dat v kernel 2 je vizualizován v obrázku 7.



Obrázek 7 Vizualizace části výpočtu na GPU (kernel 2).

6.5 Implementace na dvou GPU

Implementaci na dvou GPU lze provést mnoha způsoby. Zvolil jsem způsob, který je založen na implementaci kernelu 1. Kolekce vektorů C se zkopíruje do obou grafických karet. Kolekce jedinců z PSO P se rozdělí na polovinu a každá část se pošle na jednu GPU. Tímto jednoduchým způsobem se podařilo zkrátit výpočet zhruba na polovinu. Pro většinu vstupů je rychlost výpočtu na dvou GPU zhruba dvojnásobná oproti rychlosti výpočtu na kernelu 1. Bylo zde využito nejnovějšího přístupu k práci s více GPU. Nejnovější verze SDK umožňuje využívat najednou více GPU, bez nutnosti vytvářet vlákna v CPU.

6.6 Vytvoření výkonnostních testů

Pro jednoduché otestování výkonu jsem vytvořil možnost spustit pouze testy výkonu jednotlivých implementací (tedy test kernelu 1, kernelu 2 a použití dvou GPU). Konfigurace je možná pomocí textového souboru. Výkonnostní testy se spouštějí pomocí parametru příkazového řádku „performanceTests“. Dále se programu předá cesta ke konfiguračnímu souboru. V tabulce 10 je vidět popis konfiguračního souboru pro výkonností test. Místo znaků XXX se zadá příslušný název funkce, kterou chceme otestovat.

Název parametru	**Typ**	**Popis**
RUN_XXX	Boolean	Spustí nebo nespustí test.
DIM_XXX	Integer	Dimenze.
COL_COUNT_XXX	Integer	Počet vektorů v kolekci.
PAR_COUNT_XXX	Integer	Počet jedinců v PSO.
MAX_PAR_XXX	Integer	Maximální počet jedinců současně počítaných na GPU.
THREADS_XXX	Integer	Počet vláken v bloku.
USE_CPU_XXX	Boolean	Spustí/nespustí srovnávací test na CPU
PRECISION_XXX	Double	Nastavení přesnosti při srovnávání výsledků CPU/GPU.
REPEAT_GPU_XXX	Integer	Počet opakování testu na GPU.
REPEAT_CPU_XXX	Integer	Počet opakování testu na CPU.
WARMUP_XXX	Integer	Použití/nepoužití tzv. warmup, neboli rozehřátí, aby GPU optimalizovalo volání kernelu před samotným testováním.

Tabulka 10 Popis konfigurace výkonnostních testů.

6.7 Vytvořený program a ukládání výsledků

Vytvořil jsem aplikaci, která na základě konfigurace provede buď natrénování klasifikátoru pomocí algoritmu PSO na GPU, nebo otestuje a vyhodnotí existující vektory. Pro trénování se programu předá parametr příkazového řádku „trainPso“ a název konfiguračního souboru. Základní popis konfiguračního souboru je v tabulce 11. Podobně tak, pro testování už existujících natrénovaných vektorů se pro-

gram spustí s parametrem „testVectors“ a předá se mu cesta ke konfiguračnímu souboru. Popis konfiguračních parametrů pro testování (a validaci) vektorů je v tabulce 12. Výstupy programu jsou jednak do konzole a jednak se ukládají do souborů, ze kterých se mohou jednoduše zpracovat, nebo importovat například do programu Excel.

Název parametru	**Typ**	**Popis**
COLLECTION_FILE_NAME	String	Cesta ke kolekci.
THRESHOLDS	Integer/List	Hodnota threshold. Buď jedno číslo, pak je threshold stejný pro všechny kategorie, nebo seznam čísel oddělený středníkem, pak každá kategorie má svůj threshold.
HISTORY_FILE_NAME	String	Název souboru, kde se má ukládat průběh učení.
HISTORY_SUM_FILE_NAME	String	Název souboru, kde se má ukládat sumarizovaná hodnota o průběhu učení.
RESULT_VECTORS	String	Název souboru, kde se mají ukládat výsledné vektory.
PSO_PARTICLES_COUNT	Integer	Počet jedinců.
PSO_MAX_ITER	Integer	Maximální počet iterací.
PSO_MAX_VELOCITY	Double	Maximální rychlost částic.
PSO_K	Double	Parametr k algoritmu PSO.
PSO_C1	Double	Parametr c1 algoritmu PSO.
PSO_C2	Double	Parametr c2 algoritmu PSO.
THREADS_SIZE	Integer	Počet vláken v bloku.
USE_GPU	Integer	Možné hodnoty jsou: 0-použít jen CPU, 1-použít GPU kernel 1, 2-použít GPU kernel 2.
USE_2GPUs	Boolean	Použít/nepoužít dvě GPU.

Tabulka 11 Popis konfigurace pro trénování PSO.

Název parametru	**Typ**	**Popis**
COLLECTION_FILE_NAME	String	Cesta k souboru, kde je testovací kolekce vektorů.
TRAINED_VECTORS_FILE_NAME	String	Cesta k natrénovaným vektorům.
TESTING_HISTORY_FILE_NAME	String	Jméno souboru, kde se má ukládat historie testování.

Tabulka 12 Popis konfigurace testování/validace vektorů.

7 Použité testovací kolekce

7.1 Testovací kolekce Reuters

Testovací kolekce Reuters-22173 a později Reuters-21578, vznikla v roce 1987 a je v současné době stále nejpoužívanější a nejznámější. V minulosti chyběla standardizovaná kolekce a bylo problematické srovnávat výsledky výzkumů. Kolekce Reuters byla vytvořena za účelem porovnání nejrůznějších výzkumů například v Information retrieval, nebo Machine learning.

Skládá se z článků, které byly manuálně roztříděny do více než sta kategorií a později byly převedeny do XML. V roce 1996 Steve Finch a David D. Lewis provedli odstranění opakujících se 595 článků a další vylepšení. Z kolekce Reuters-22173 tak vznikla kolekce Reuters-21578, která se stala novým standardem. Informace v této kapitole jsou čerpány z dokumentu [7], který slouží jako dokumentace ke kolekci Reuters-22173 a je přiložen v této kolekci.

7.1.1 Struktura kolekce

Takto vypadá náhodně vybraný článek z této kolekce:

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OL-
DID="14533" NEWID="9012">
<DATE>24-MAR-1987 16:07:51.39</DATE>
<TOPICS><D>interest</D></TOPICS>
<PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;A RM
&#22;&#22;&#1;f2443&#31;reute
u f BC-TREASURY-BALANCES-AT   03-24 0084</UNKNOWN>
<TEXT>&#2;
<TITLE>TREASURY BALANCES AT FED ROSE ON MARCH 23</TITLE>
<DATELINE>     WASHINGTON, March 24 - </DATELINE><BODY>Treasury ba-
lances at the Federal
Reserve rose on March 23 to 3.332 billion dlrs from 3.062
billion dlrs on the previous business day, the Treasury said in
its latest budget statement.
    Balances in tax and loan note accounts fell to 15.513
billion dlrs from 17.257 billion dlrs on the same respective
days.
    The Treasury's operating cash balance totaled 18.845
billion dlrs on March 23 compared with 20.318 billion dlrs on
```

```
March 20.
 Reuter
&#3;</BODY></TEXT>
</REUTERS>
```

Celá kolekce je distribuována v 21 souborech, kde každý kromě posledního obsahuje 1000 článků. Jednotlivé články začínají tagem *<REUTERS TOPICS=?? LEWISSPLIT=?? CGISPLIT=?? OLDID=?? NEWID=?? >* a končí *</REUTERS>*. Zde je popis atributů tohoto tagu:

TOPICS – Může nabývat hodnoty *YES*, *NO* a *BYPASS*. Uchovává informaci, zdali předchozí kolekce Reuters-22173 obsahovala v tagu *<TOPICS>* nějaké kategorie. Je třeba zmínit, že hodnota tohoto atributu nemá žádnou vazbu na kolekci Reuters-21578.

LEWISSPLIT – Možné hodnoty jsou *TEST*, *TRAINING*, *NOT-USED*. Znamenají, jak David D. Lewis použil konkrétní článek ve svých experimentech. Pokud článek zahrnul do testovací kolekce, má označení *TEST*, pokud ho zahrnul do trénovací kolekce, má označení TRAINING. Články, které nepoužil, označil *NOT-USED*.

CGISPLIT – Je další způsob rozdělení kolekce na trénovací (*TRAINING-SET*) a testovací (*PUBLISHED-TESTSET*) dokumenty. Tento způsob rozdělení použil ve svých experimentech Philip J. Hayes.

OLDID – Identifikační číslo článku v kolekci Reuters-22173.

NEWID – Nové identifikační číslo používané v Reuters-21578.

CSECS – Objevuje se jen někde, může být ignorován.

Z vnitřních tagů jsou pro kategorizaci dokumentů důležité jen tyto:

<TOPICS> – Uchovává informaci, do jakých kategorií článek spadá a může být i prázdný. Jednotlivé kategorie jsou ještě ohraničeny tagem *<D>*.

<TEXT> – Odděluje veškerý textový obsah článku. Uvnitř něho jsou další tagy.

<TITLE> – Titulek článku. Pokud má tag *<TEXT>* atribut *TYPE="BRIEF"*, tak je v tomto elementu celý článek typu „stručný".

<BODY> – Hlavní text článku.

7.1.2 Použití kolekce pro kategorizaci textů

V kolekci je celkem 135 kategorií. Kategorií, které mají alespoň jeden dokument, je 120. Kategorií, které se vyskytují alespoň ve 20 dokumentech, je 57. Mnoho kategorií se tedy neobjeví v žádném dokumentu. Při hodnocení nového klasifikátoru se ale doporučuje, zahrnout i tyto nikdy nevyužité kategorie. Také je důležité, že znalosti vlastností a charakteristik testovaných dat, by neměla mít vliv na vývoj systému, protože by byl výkon takového systému zkreslený.

Výsledky efektivity kategorizování se mohou porovnávat jen mezi studiemi, které mají stejné množiny trénovacích a testovacích dokumentů. Starší verze Reuters-22173 měla nejednoznačné formátování. Kvůli nejasné interpretaci někteří autoři například vynechali články, které neměly žádnou kategorii. Ve verzi Reuters-21578 je přesně specifikováno, jaké rozdělení je doporučené použít.

7.1.3 Možné množiny trénovacích a testovacích dokumentů

ModLewis:

- Trénovací množina má 13625 článků. Ty musí mít atributy *LEWISSPLIT="TRAIN", TOPICS="YES"*, nebo *"NO"*.
- Testovací množina má 6188 článků, *LEWISSPLIT="TEST"*, *TOPICS="YES"*, nebo *"NO"*
- Nepoužitých článků je 1765 a ty mají atributy *LEWISSPLIT="NOT-USED"*, nebo *TOPICS="BYPASS"*.
- Nepoužité články mohou sloužit například pro získání statistických informací o distribuci slov.
- Toto rozdělení dokumentů se liší u starší a novější verze Reuters, proto výsledky nemohou být mezi sebou srovnávány.

ModApte:

- Trénovací množina má 9603 článků, atributy článků v této množině musí nabývat hodnot *LEWISSPLIT="TRAIN"* a *TOPICS="YES"*.
- Testovací množina má 3299 článků, *LEWISSPLIT="TEST"* a *TOPICS="YES"*.
- Nepoužitých je 8676 článků s atributy *LEWISSPLIT="NOT-USED"*, *TOPICS="YES",* nebo *TOPICS="NO"*, nebo *TOPICS="BYPASS"*.
- Nepoužité články mohou stejně jako u ModLewis například sloužit pro získávání statistických informací o slovech.
- Stejně jako ModLewis nelze srovnávat výsledky mezi starší a novější verzí Reuters.
- Toto rozdělení dokumentů je nejvíce doporučováno.

ModHayes:

- Trénovací množina má 20856 článků, *CGISPLIT="TRAINING-SET"*.
- Testovací množina má 722 článků, *CGISPLIT="PUBLISHED-TESTSET"*.
- Všechny články jsou tedy použité.
- Pokud se na toto rozdělení aplikuje ještě další sada úprav, které jsou blíže popsány v [7], tak je toto rozdělení článků srovnatelné se starší verzi Reuters-22173.

7.1.4 Jiná rozdělení

Je obecně doporučováno nevymýšlet vlastní rozdělování množin dokumentů, které je založeno na tom, jestli mají články nějaký název kategorie. Dále se doporučuje neprovádět testy jen na „jednoduchých“ kategoriích. Některé ze 135 kategorií totiž nemají žádné, nebo mají málo pozitivních trénovacích příkladů. Některé nemají žádné pozitivní testovací příklady nebo jich mají málo. Některé mají málo jak

pozitivních trénovacích, tak pozitivních testovacích příkladů. V [4] autoři použili jen 10 kategorií s největší četností dokumentů.

7.2 Testovací kolekce Iris

Kolekce Iris nebo také Fisher's Iris data set vznikla už v roce 1936 je stažitelná z [14]. Data obsahují tři kategorie, kde každá obsahuje padesát měřených dat. Měřené data se týkají rostliny zvané Kosatec a skládají ze čtyř atributů (délka kalichu, šířka kalichu, délka a šířka okvětního lístku). Klasifikace se týká rozeznávání mezi třemi druhy této rostliny (Iris Setosa, Iris Versicolour a Iris Virginica). Jedna třída je lineárně separovatelná, další dvě nejsou [14]. Tato kolekce samozřejmě nesouvisí s klasifikací textů, ale chtěl jsem demonstrovat na jednoduchém příkladu, že vytvořený program lze použít na libovolnou kolekci. Stačí mít data roztříděna v kategoriích a převedena do vektorů. Příklad několika rostlin a jejich atributů je zde:

Iris Setosa				**Iris Versicolour**				**Iris Virginica**			
Kalich		**Okvětní lístek**		**Kalich**		**Okvětní lístek**		**Kalich**		**Okvětní lístek**	
Délka	**Šířka**	**Délka**	**Šířka**	**Délka**	**Šířka**	**Délka**	**Šířka**	**Délka**	**Šířka**	**Délka**	**Šířka**
5,1	3,5	1,4	0,2	7	3,2	4,7	1,4	6,3	3,3	6	6,3
4,9	3	1,4	0,2	6,4	3,2	4,5	1,5	5,8	2,7	5,1	5,8
4,7	3,2	1,3	0,2	6,9	3,1	4,9	1,5	7,1	3	5,9	7,1
4,6	3,1	1,5	0,2	5,5	2,3	4	1,3	6,3	2,9	5,6	6,3
5	3,6	1,4	0,2	6,5	2,8	4,6	1,5	6,5	3	5,8	6,5
5,4	3,9	1,7	0,4	5,7	2,8	4,5	1,3	7,6	3	6,6	7,6
4,6	3,4	1,4	0,3	6,3	3,3	4,7	1,6	4,9	2,5	4,5	4,9
5	3,4	1,5	0,2	4,9	2,4	3,3	1	7,3	2,9	6,3	7,3
4,4	2,9	1,4	0,2	6,6	2,9	4,6	1,3	6,7	2,5	5,8	6,7
4,9	3,1	1,5	0,1	5,2	2,7	3,9	1,4	7,2	3,6	6,1	7,2

Tabulka 13 Ukázka dat z kolekce Iris.

7.3 Kolekce 20 Newsgroup

Tato kolekce je volně stažitelná z [13]. Obsahuje přibližně dvacet tisíc dokumentů (e-mailů) rozdělených do dvaceti kategorií. Některé kategorie jsou si blízké (např. comp.sys.ibm.pc.hardware / comp.sys.mac.hardware) a některé jsou si vzdálené (např. misc.forsale / soc.religion.christian). Existují tři hlavní verze. První verze „19997" je originální a nemodifikovaná verze. Druhá verze „bydate" je seřazena podle data a má 18846 dokumentů. Obsahuje šedesát procent trénovacích a čtyřicet procent testovacích dokumentů a neobsahuje duplikáty. Třetí „18828" také neobsahuje duplikáty, ale obsahuje jen hlavičky „From" a „Subject". V [13] je pro experimenty doporučeno použít druhou verzi se zdůvodněním, že je jednodušší srovnávat různé experimenty, protože se ve výběru testovacích a

trénovacích příkladů neobjevuje náhoda. Data jsou totiž seřazena podle data a testovací množina tak simuluje nárůst dokumentů v čase. Zde je příklad náhodně vybraného dokumentu z této kolekce:

```
From: markg@sr.hp.com (Mark Goldsworthy)
Subject: Re: Ad said Nissan Altima best seller?
Organization: HP Sonoma County (SRSD/MWTD/MID)
X-Newsreader: TIN [version 1.1 PL8.8]
Lines: 21

Spiros Triantafyllopoulos (c23st@kocrsv01.delcoelect.com) wrote:
: But waiiiiiit, isn't Nissan officially registering the car as far
as
: government paperwork goes, Nissan Stanza Altima, to avoid costly
and
: lengthy paperwork? I read this on the net a while ago, and someone
: actually may have said there's a little Stanza logo on the Altima
: somewhere.

I just bought an Altima (and like it very much) and yes there is a
little Stanza logo ever so discretely placed on the trunk. The Alti-
ma is
emblazoned in big silver letters, but the itsy-bitsy Stanza is shun-
ted
to the far left of the trunk lid. You can only see it if you get up
close to the car and know where to look. It is very inconspicuous.

In fact my first clue that this was a Stanza was that the owners ma-
nual
called the car a Nissan Stanza Altima.

Anybody know *why* Nissan did it this way?

Mark Goldsworthy
```

8 Experimenty a dosažené výsledky

Byly provedeny jak experimenty na výše popsaných kolekcích, tak výkonnostní testy. Je také třeba zmínit, že všechny testy jsou časově velmi náročné. I přes zrychlení, kterého se podařilo dosáhnout, jeden experiment může trvat od několika minut až po několik hodin. Pro každou kategorii je nutné provést okolo deseti testů, aby bylo možné vyladit parametr threshold. Testovaná kolekce Reuters má po předzpracování deset, kolekce Newsgroup pak dvacet kategorií. Když jsou provedeny testy pro další dimenze (například 300 a 1000), nebo jinou konfiguraci, znamená to samozřejmě nové testování a ladění parametrů. Kolekce Iris (150 prvků dimenze 4) byla z testovaných kolekcí nejmenší a její náročnost při testování se pohybovalo v řádu minut. Stejně tak výkonnostní testy se pohybovaly v řádu minut. Nejdelší trval zhruba jen devadesát minut. Všechny výkonnostní testy byly provedeny třikrát a spočten jejich aritmetický průměr.

U kolekcí Reuters a Newsgroup byly při předzpracování odstraněny všechny slova kratší než tři znaky. Byla odstraněna všechna čísla, slova převedena na malá písmena. Dále byla odstraněna stop slova (kapitola 1.2). Pro převedení slov na základní tvar byl použit algoritmus Porter Stemmer (kapitola 1.3). Pro všechny testy byla z trénovací množiny oddělena třetina dokumentů, která posloužila pro validaci a ladění threshold. Jedinou výjimkou byla 3-fold cross validace použita pro testování kolekce Reuters. Ta má odlišný přístup (více v kapitole 2.3). Transformace dokumentů na vektory byla provedena podle kapitoly 1.4 a 1.5). Pro naučení vektorů pro algoritmus PSO použito 10000 jedinců, 15 iterací. Parametr c_1=2,0, c_2=2,0, k=0,99 a rychlost byla omezena na hodnotu 25. Každý z těchto parametrů samozřejmě může mít vliv na kvalitu výsledku.

8.1 Experimenty na kolekci Reuters

Na kolekci Reuters byly provedeny celkem čtyři větší experimenty. Z kolekce bylo vybráno 10 kategorií s největším počtem dokumentů, stejně jako v [4]. Byly ignorovány všechny dokumenty, které obsahují jiné kategorie, než těchto deset. Experimenty byly pojmenovány R-300, R-ma-300, R-ma-1000 a R-3fold-300. Písmeno „R“ odpovídá kolekci Reuters, „ma“ znamená standardní rozdělení „modApte“ (kapitola 7.1), 300 nebo 1000 jsou dimenze vektorů a „3fold“ je 3-fold cross validace (kapitola 2.3).

8.1.1 Experiment R-300

Při tomto experimentu byly z kolekce Reuters vyjmuty dokumenty patřící do deseti největších kategorií a náhodně zamíchány. Zhruba šedesát procent z těchto dokumentů bylo vloženo do trénovací, dvacet do validační a zbylých dvacet procent do testovací množiny. Nejdříve bylo provedeno celkem 13 testů pro 13 různých threshold od 0,0 do 0,3 s krokem 0,025. Tyto hodnoty jsou zobrazeny v tabulce 14. Nejvyšší hodnota F_1 pro daný threshold (tabulka 15) pro každou kategorii byla použita pro nové natrénování na trénovací a následné testování na testovací množině dokumentů. V tabulce 17 je pak vidět finální micro a macro F_1 skóre spočítané pro všechny kategorie na testovací množině.

threshold	wheat	acq	trade	ship	interest	grain	crude	earn	money-fx	corn
0	0,6667	0,8741	0,7657	0,5455	0,7027	0,8154	0,7393	0,9379	0,6106	0,3103
0,025	0,9143	0,8552	0,8098	0,6667	0,7914	0,7778	0,8000	0,9389	0,7435	0,4444
0,05	0,8710	0,8747	0,8333	**0,7692**	0,7525	0,8182	0,8235	0,9452	0,7353	0,5333
0,075	0,8529	**0,8956**	0,8228	0,6667	**0,8021**	0,8065	0,8023	**0,9504**	**0,7897**	0,5417
0,1	0,9091	0,8399	0,8535	0,7255	0,7419	0,8125	0,7910	0,9419	0,7660	0,6000
0,125	**0,9375**	0,8410	0,8466	0,7184	0,7241	0,8235	0,8140	0,8816	0,6404	0,5714
0,15	0,9375	0,7902	0,8442	0,6875	0,7657	**0,8571**	0,8095	0,9317	0,7135	0,3902
0,175	0,9231	0,7862	**0,8742**	0,6667	0,7558	0,8480	**0,8571**	0,8987	0,6905	0,6250
0,2	0,8571	0,7335	0,8435	0,6957	0,7394	0,8000	0,7879	0,8841	0,7160	**0,6383**
0,225	0,9032	0,5966	0,8378	0,6154	0,7738	0,7895	0,8000	0,7837	0,7485	0,5957
0,25	0,9032	0,5073	0,8252	0,6304	0,6962	0,7706	0,7682	0,7007	0,6323	0,5455
0,275	0,8475	0,4071	0,7971	0,6437	0,5882	0,7636	0,6466	0,6542	0,5714	0,4211
0,3	0,8475	0,2051	0,6772	0,5610	0,6846	0,7407	0,5124	0,5662	0,5037	0,4103

Tabulka 14 Výběr/ladění threshold pro experiment R-300.

	wheat	acq	trade	ship	interest	grain	crude	earn	money-fx	corn
threshold	**0,125**	**0,075**	**0,175**	**0,050**	**0,075**	**0,150**	**0,175**	**0,075**	**0,075**	**0,20**

Tabulka 15 Nejlepší threshold pro experiment R-300.

	threshold	TP	FP	FN	Precision	Recall	F1
wheat	0,1250	56	5	3	0,9180	0,9492	0,9333
acq	0,0750	404	54	58	0,8821	0,8745	0,8783
trade	0,1750	63	9	15	0,8750	0,8077	0,8400
ship	0,0500	31	18	15	0,6327	0,6739	0,6526
interest	0,0750	65	18	24	0,7831	0,7303	0,7558
grain	0,1500	78	2	21	0,9750	0,7879	0,8715
crude	0,1750	93	10	20	0,9029	0,8230	0,8611
earn	0,0750	766	31	72	0,9611	0,9141	0,9370
money-fx	0,0750	77	12	21	0,8652	0,7857	0,8235
corn	0,2000	12	5	17	0,7059	0,4138	0,5217

Tabulka 16 Výsledky F1 skóre pro jednotlivé kategorie (R-300).

	Micro	Macro
Precision	0,9093	0,8501
Recall	0,8608	0,7760
F1	0,8844	0,8114

Tabulka 17 F1 skóre pro R-300.

8.1.2 Experiment R-ma-300

Tento experiment testuje kolekci na standardním rozdělení „modApte“ (kapitola 7.1.3). Toto rozdělení nepočítá s validační množinou. Z trénovací množiny tak bylo vyjmuto 30 procent náhodně vybraných dokumentů a ty byly použity jako validační množina pro ladění threshold. V tabulce 18 jsou opět červeně vyznačené nejlepší hodnoty F skóre. Tabulka 19 souhrnně ukazuje nejlepší threshold pro každou kategorii. Výsledné hodnoty jsou v tabulkách 20 a 21.

threshold	wheat	acq	trade	ship	interest	grain	crude	earn	money-fx	corn
0	0,8000	0,8416	0,7254	0,5541	0,6783	0,6740	0,7091	0,9262	0,5930	0,4179
0,025	**0,9412**	**0,8718**	0,7636	0,6552	0,7009	0,7421	0,7802	0,9187	0,6923	0,4828
0,05	0,9286	0,8685	0,7841	**0,7290**	0,7429	0,7237	0,8432	0,9359	0,7254	0,4528
0,075	0,9136	0,8704	**0,8202**	0,6327	0,7035	0,6974	0,8541	**0,9428**	0,7340	0,5000
0,1	0,9114	0,8591	0,8148	0,6667	0,7150	0,7361	0,8265	0,9185	0,7200	0,4815
0,125	0,9136	0,8202	0,8136	0,6875	0,7368	**0,7848**	0,8667	0,9269	0,6941	0,4255
0,15	0,8974	0,7966	0,7952	0,6667	**0,7822**	0,7632	0,8229	0,9043	**0,7458**	0,4889
0,175	0,8974	0,7974	0,7821	0,7129	0,6911	0,7310	**0,8764**	0,8513	0,7081	0,4681
0,2	0,8831	0,6588	0,7643	0,6869	0,7644	0,7133	0,8736	0,7847	0,7607	0,4583
0,225	0,9268	0,6018	0,7651	0,6667	0,6977	0,7092	0,8621	0,7076	0,6918	**0,5106**
0,25	0,9114	0,3580	0,7133	0,6022	0,7263	0,7050	0,8372	0,7082	0,6620	0,5106
0,275	0,8649	0,2538	0,6857	0,6047	0,5963	0,6667	0,7692	0,6076	0,5373	0,4490
0,3	0,8649	0,2471	0,7111	0,6047	0,5625	0,6212	0,7027	0,5610	0,4800	0,4706

Tabulka 18 Hledání nejlepšího threshold (R-ma-300).

	wheat	acq	trade	ship	interest	grain	crude	earn	money-fx	corn
threshold	**0,025**	**0,025**	**0,075**	**0,05**	**0,15**	**0,125**	**0,175**	**0,075**	**0,15**	**0,225**

Tabulka 19 Nejlepší nalezené threshold na validační množině (R-ma-300).

	threshold	TP	FP	FN	Precision	Recall	F1
wheat	0,025	37	11	3	0,7708	0,9250	0,8409
acq	0,025	635	56	65	0,9190	0,9071	0,9130
trade	0,075	63	33	20	0,6563	0,7590	0,7039
ship	0,050	58	18	23	0,7632	0,7160	0,7389
interest	0,150	83	19	37	0,8137	0,6917	0,7477
grain	0,125	57	4	22	0,9344	0,7215	0,8143
crude	0,175	145	18	17	0,8896	0,8951	0,8923
earn	0,075	979	75	107	0,9288	0,9015	0,9150
money-fx	0,150	65	20	65	0,7647	0,5000	0,6047
corn	0,225	13	13	12	0,5000	0,5200	0,5098

Tabulka 20 Výsledné hodnoty pro všechny kategorie (R-ma-300).

	Micro	Macro
Precision	0,8888	0,7940
Recall	0,8520	0,7537
F1	0,8700	0,7733

Tabulka 21 Celkové F1 skóre pro experiment R-ma-300.

8.1.3 Experiment R-3fold-300

Experiment R-3fold-300 je zajímavý hned ze dvou hledisek. Jednak tím, že je zde použita k-fold cross validace, konkrétně 3-fold cross validace a jednak tím, že se podobný experiment vyskytuje v publikaci [4]. Autoři zde bohužel neuvedli úplné konfigurace parametrů, ale pro zběžné srovnání to stačí. Na rozdíl od jiných experimentů, u metody měření 3-fold cross není dostupná validační množina. Byly provedeny celkem tři měření pro každý ze třinácti threshold na trénovací množině. Protože chceme najít nejlepší threshold takový, který by se co nejlépe choval i pro nové dokumenty, tedy v našem případě testovací množinu, nemá smysl určovat jiný threshold pro každý „fold“ zvlášť. F1 hodnoty všech tří měření byly zprůměrované (tabulka 22). Byl vybrán nejlepší threshold pro každou kategorii (tabulka 23). V tabulce 24 jsou vidět výsledky pro jednotlivé kategorie. V tabulce 25 je vidět celkové vyhodnocení pro všechny kategorie dohromady.

threshold	wheat	acq	trade	ship	interest	grain	crude	earn	money-fx	corn
0	0,8720	0,8202	0,7442	0,7022	0,6131	0,7494	0,7745	0,9335	0,6853	0,4297
0,025	0,8640	0,9120	0,7687	0,7268	0,6805	0,7708	0,8196	0,9374	0,6777	0,4607
0,05	**0,8888**	0,9159	0,8263	0,7634	0,7142	0,7823	0,8412	0,9421	0,7650	0,5051

0,075	0,8782	**0,9421**	**0,8337**	**0,7939**	0,7140	0,8059	0,8313	0,9287	0,7544	0,4596
0,1	0,8704	0,9209	0,8136	0,7481	**0,7471**	0,8084	0,8508	**0,9467**	0,7627	**0,5485**
0,125	0,8660	0,9308	0,8302	0,7925	0,7295	0,8084	0,8493	0,9320	0,7683	0,5031
0,15	0,8272	0,9252	0,8197	0,7563	0,7205	**0,8150**	**0,8600**	0,9110	**0,7787**	0,4896
0,175	0,7715	0,9188	0,8336	0,7762	0,7382	0,8145	0,8547	0,8613	0,7085	0,5220
0,2	0,6553	0,8990	0,7957	0,7425	0,6998	0,7866	0,8282	0,8263	0,7478	0,5372
0,225	0,5493	0,8992	0,7883	0,7045	0,6795	0,7722	0,8446	0,7313	0,7199	0,5000
0,25	0,4788	0,9001	0,8125	0,6958	0,6702	0,7514	0,7895	0,6924	0,6095	0,4829
0,275	0,2516	0,8957	0,7307	0,6692	0,6416	0,7431	0,7523	0,6367	0,5727	0,4925
0,3	0,1738	0,8920	0,6806	0,6222	0,6034	0,6944	0,6741	0,5213	0,4485	0,4540

Tabulka 22 Hledání nejlepšího threshold (R-3fold-300)

	acq	**wheat**	**trade**	**interest**	**ship**	**grain**	**crude**	**earn**	**money-fx**	**corn**
threshold	**0,050**	**0,075**	**0,075**	**0,075**	**0,100**	**0,150**	**0,150**	**0,100**	**0,150**	**0,100**

Tabulka 23 Nejlepší nalezená threshold (R-3fold-300)

	threshold	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1**
acq	0,050	675	84	126	0,8889	0,8431	0,8649
wheat	0,075	62	5	4	0,9192	0,9479	0,9328
trade	0,075	108	23	25	0,8270	0,8159	0,8210
interest	0,075	120	33	32	0,7823	0,7879	0,7844
ship	0,100	57	13	34	0,8201	0,6251	0,7084
grain	0,150	95	9	32	0,9140	0,7468	0,8217
crude	0,150	137	16	33	0,8940	0,8075	0,8480
earn	0,100	1226	69	97	0,9468	0,9266	0,9364
money-fx	0,150	119	29	49	0,8111	0,7071	0,7523
corn	0,100	21	15	23	0,6052	0,4959	0,5287

Tabulka 24 Výsledky testování jednotlivých kategorií (R-3fold-300).

	Micro	Macro
Precision	0,8984	0,8409
Recall	0,8518	0,7704
F1	0,8745	0,8041

Tabulka 25 Celkové výsledky všech kategorií (R-3fold-300).

8.1.4 Experiment R-ma-1000

Experiment je stejný jako R-ma-300 s tím rozdílem, že dimenze vektorů je nastavena na hodnotu 1000. Tabulka 26 zobrazuje ladění threshold, tabulka 27 jeho nejlepší hodnoty pro každou kategorii, tabulka 28 a 29 pak výsledné hodnoty.

threshold	wheat	acq	trade	ship	interest	grain	crude	earn	money-fx	corn
0,000	0,5895	0,7926	0,6990	0,6296	0,6400	0,6994	0,5319	0,8944	0,5789	0,4810
0,025	0,9041	**0,7991**	0,7834	0,6422	0,6392	0,7468	0,7607	**0,9170**	0,6883	0,6667
0,050	**0,9744**	0,7183	0,8043	0,7727	**0,7129**	0,7785	0,7237	0,9088	**0,7830**	0,6667
0,075	0,9189	0,7346	**0,8394**	0,7111	0,6943	0,8435	**0,8158**	0,8821	0,7500	**0,7797**
0,100	0,9315	0,6538	0,8333	0,7033	0,6474	**0,8767**	0,7286	0,8172	0,7685	0,6531
0,125	0,8767	0,5258	0,7614	**0,7907**	0,7089	0,8529	0,6812	0,7523	0,7347	0,7308
0,150	0,8986	0,2128	0,7692	0,6053	0,6250	0,7344	0,6765	0,6290	0,6171	0,6923
0,175	0,8485	0,1633	0,6667	0,5429	0,3443	0,5556	0,5500	0,3732	0,3404	0,6667
0,200	0,8657	0,0387	0,3175	0,4545	0,6752	0,3960	0,2105	0,4162	0,2500	0,6667
0,225	0,8125	0,0921	0,4741	0,0370	0,2879	0,5283	0,4386	0,4895	0,2656	0,6667
0,250	0,7937	0,0844	0,0571	0,2105	0,1940	0,5000	0,1720	0,3432	0,1746	0,4545
0,275	0,0513	0,0217	0,1947	0,2712	0,0762	0,4615	0,1720	0,3009	0,1719	0,5217
0,300	0,7541	0,0513	0,0571	0,3000	0,2017	0,0000	0,0444	0,1655	0,4521	0,5532

Tabulka 26 Hledání threshold (R-ma-1000).

	acq	wheat	trade	interest	ship	grain	crude	earn	money-fx	corn
threshold	**0,050**	**0,075**	**0,075**	**0,075**	**0,100**	**0,150**	**0,150**	**0,100**	**0,150**	**0,100**

Tabulka 27 Nejlepší nalezená threshold (R-ma-1000).

Kategorie	Threshold	TP	FP	FN	Precision	Recall	F1
wheat	0,050	39	5	1	0,8864	0,9750	0,9286
acq	0,025	549	160	151	0,7743	0,7843	0,7793
trade	0,075	66	16	17	0,8049	0,7952	0,8000
ship	0,125	36	4	45	0,9000	0,4444	0,5950
interest	0,050	68	13	52	0,8395	0,5667	0,6766
grain	0,100	66	3	13	0,9565	0,8354	0,8919
crude	0,075	145	30	17	0,8286	0,8951	0,8605
earn	0,025	1009	79	77	0,9274	0,9291	0,9282
money-fx	0,050	89	45	41	0,6642	0,6846	0,6742
corn	0,075	15	0	10	1,0000	0,6000	0,7500

Tabulka 28 Výsledné hodnoty pro každou kategorii zvlášť (R-ma-1000).

	Micro	Macro
Precision	0,8543	0,8582
Recall	0,8308	0,7510
F1	0,8424	0,8010

Tabulka 29 Celkové výsledky pro všechny kategorie (R-ma-1000).

8.2 Experimenty na kolekci 20 Newsgroup

Na této kolekci byl proveden experiment označen jako NG-300. Písmena NG znamenají kolekci 20 Newsgroup a číslo 300 znamená dimenzi. Další experiment označený jako NG-1000 má dimenzi omezenou na hodnotu 1000. Byla otestována vždy celá kolekce bez omezení na konkrétní kategorie. Postup testování byl stejný jako u předešlé kolekce. Nejprve bylo provedeno několik experimentů s fixní hodnotou threshold pomocí trénovací množiny. Vyhodnocení proběhlo pomocí validační množiny. Threshold s nejlepší hodnotou účelové funkce F_1 byly použity pro naučení na trénovací množině. V tabulce jsou zvýrazněny nejlepší hodnoty. Takto naučené vektory byly otestovány na testovací množině.

8.2.1 Experiment NG-300

threshold	0,000	0,025	0,050	0,075	0,100	0,125	0,150	0,175	0,200
rec.motorcycles	0,213	0,214	0,176	0,201	0,195	0,196	0,232	0,237	**0,251**
comp.windows.x	0,343	0,287	0,319	**0,371**	0,332	0,344	0,356	0,289	0,330
talk.politics.mideast	0,289	0,301	0,323	0,345	0,300	0,376	**0,380**	0,351	0,361
talk.politics.guns	**0,317**	0,249	0,234	0,301	0,214	0,241	0,269	0,261	0,194

talk.religion.misc	0,255	0,207	0,249	0,263	0,308	0,238	0,257	**0,278**	0,233
rec.autos	0,632	0,667	0,681	0,672	0,683	0,671	**0,690**	0,655	0,690
rec.sport.baseball	0,341	0,349	0,377	**0,383**	0,361	0,327	0,358	0,376	0,265
rec.sport.hockey	0,493	0,497	**0,505**	0,496	0,517	0,475	0,497	0,475	0,497
comp.sys.mac.hardware	**0,256**	0,245	0,244	0,157	0,201	0,154	0,186	0,164	0,217
sci.space	**0,540**	0,513	0,511	0,525	0,514	0,516	0,509	0,513	0,506
comp.sys.ibm.pc.hardware	0,311	0,353	0,364	0,339	0,372	**0,400**	0,400	0,370	0,354
talk.politics.misc	0,231	0,206	0,331	0,252	0,311	**0,331**	0,254	0,303	0,279
comp.graphics	0,244	0,268	0,256	0,290	0,274	0,352	**0,354**	0,288	0,272
sci.electronics	**0,260**	0,246	0,164	0,236	0,188	0,211	0,201	0,202	0,184
soc.religion.christian	0,555	0,599	0,604	0,618	0,586	0,551	0,568	**0,623**	0,578
sci.med	0,225	0,231	0,251	0,286	**0,332**	0,286	0,249	0,269	0,237
sci.crypt	0,564	0,609	0,545	0,585	**0,601**	0,595	0,592	0,595	0,580
alt.atheism	0,225	0,288	0,297	0,303	**0,306**	0,279	0,261	0,249	0,248
misc.forsale	0,635	**0,673**	0,626	0,639	0,629	0,662	0,637	0,640	0,646
comp.os.ms-windows.misc	0,540	**0,582**	0,562	0,565	0,580	0,568	0,541	0,555	0,534

Tabulka 30 Výběr threshold (NG-300)

Kategorie	threshold	TP	FP	FN	Precision	Recall	F1
rec.motorcycles	0,200	201	689	441	0,2258	0,3131	0,2624
comp.windows.x	0,000	147	528	248	0,2178	0,3722	0,2748
talk.politics.mideast	0,150	160	354	281	0,3113	0,3628	0,3351
talk.politics.guns	0,000	112	409	298	0,2150	0,2732	0,2406
talk.religion.misc	0,175	74	280	177	0,2090	0,2948	0,2446
rec.autos	0,150	212	117	184	0,6444	0,5354	0,5848
rec.sport.baseball	0,075	160	463	481	0,2568	0,2496	0,2532
rec.sport.hockey	0,050	264	239	446	0,5249	0,3718	0,4353
comp.sys.mac.hardware	0,000	123	494	262	0,1994	0,3195	0,2455
sci.space	0,000	201	173	193	0,5374	0,5102	0,5234
comp.sys.ibm.pc.hardware	0,125	134	256	258	0,3436	0,3418	0,3427
talk.politics.misc	0,125	78	231	232	0,2524	0,2516	0,2520
comp.graphics	0,150	110	249	279	0,3064	0,2828	0,2941
sci.electronics	0,000	216	612	481	0,2609	0,3099	0,2833

soc.religion.christian	0,175	218	201	180	0,5203	0,5477	0,5337
sci.med	0,100	93	270	303	0,2562	0,2348	0,2451
sci.crypt	0,100	167	115	229	0,5922	0,4217	0,4926
alt.atheism	0,100	93	354	357	0,2081	0,2067	0,2074
misc.forsale	0,025	307	165	148	0,6504	0,6747	0,6624
comp.os.ms-windows.misc	0,025	240	200	154	0,5455	0,6091	0,5755

Tabulka 31Výsledky jednotlivých kategorií (NG-300).

	Micro	**Macro**
Precision	0,3409	0,3639
Recall	0,3702	0,3742
F1	0,3549	0,3690

Tabulka 32 Výsledné hodnoty pro všechny kategorie (NG-300).

8.2.2 Experiment NG-1000

	0,000	**0,025**	**0,050**	**0,075**	**0,100**	**0,125**	**0,150**	**0,175**	**0,200**
rec.motorcycles	0,495	0,521	**0,711**	0,569	0,582	0,570	0,547	0,444	0,417
comp.windows.x	0,393	0,360	0,351	**0,442**	0,331	0,235	0,280	0,237	0,188
talk.politics.mideast	0,579	0,602	0,594	**0,650**	0,645	0,634	0,509	0,563	0,409
talk.politics.guns	0,520	0,665	0,651	**0,698**	0,628	0,648	0,576	0,510	0,203
talk.religion.misc	0,228	0,280	0,276	**0,316**	0,267	0,220	0,233	0,148	0,136
rec.autos	0,568	0,627	**0,640**	0,605	0,625	0,619	0,538	0,505	0,426
rec.sport.baseball	0,445	**0,461**	0,344	0,381	0,342	0,301	0,310	0,201	0,308
rec.sport.hockey	0,470	**0,585**	0,555	0,460	0,500	0,500	0,451	0,328	0,242
comp.sys.mac.hardware	0,401	**0,545**	0,511	0,504	0,422	0,480	0,448	0,057	0,113
sci.space	0,434	0,486	0,534	0,553	0,514	**0,574**	0,508	0,433	0,400
comp.sys.ibm.pc.hardware	0,380	0,344	0,332	0,375	**0,404**	0,385	0,373	0,246	0,192
talk.politics.misc	0,347	0,225	0,298	**0,423**	0,287	0,271	0,132	0,166	0,176
comp.graphics	0,417	0,400	0,414	0,452	**0,497**	0,403	0,209	0,189	0,290
sci.electronics	**0,259**	0,215	0,239	0,203	0,160	0,113	0,096	0,050	0,024
soc.religion.christian	0,518	**0,566**	0,482	0,507	0,526	0,431	0,412	0,400	0,228
sci.med	0,372	0,399	0,357	0,402	**0,407**	0,355	0,272	0,313	0,047
sci.crypt	0,593	0,675	0,716	0,704	0,688	**0,717**	0,667	0,465	0,537

alt.atheism	0,389	0,425	**0,459**	0,417	0,358	0,454	0,273	0,282	0,122
misc.forsale	0,457	0,585	**0,627**	0,606	0,570	0,506	0,586	0,505	0,067
comp.os.ms-windows.misc	0,518	0,510	0,574	0,551	**0,578**	0,512	0,458	0,459	0,406

Tabulka 33 Výběr threshold (NG-1000).

Kategorie	threshold	TP	FP	FN	Precision	Recall	F1
rec.motorcycles	0,050	267	50	375	0,8423	0,4159	0,5568
comp.windows.x	0,075	158	241	237	0,3960	0,4000	0,3980
talk.politics.mideast	0,075	155	44	286	0,7789	0,3515	0,4844
talk.politics.guns	0,075	87	103	323	0,4579	0,2122	0,2900
talk.religion.misc	0,075	37	169	214	0,1796	0,1474	0,1619
rec.autos	0,050	232	116	164	0,6667	0,5859	0,6237
rec.sport.baseball	0,025	228	250	413	0,4770	0,3557	0,4075
rec.sport.hockey	0,025	279	209	431	0,5717	0,3930	0,4658
comp.sys.mac.hardware	0,025	212	224	173	0,4862	0,5506	0,5164
sci.space	0,125	154	56	240	0,7333	0,3909	0,5099
comp.sys.ibm.pc.hardware	0,100	147	222	245	0,3984	0,3750	0,3863
talk.politics.misc	0,075	29	71	281	0,2900	0,0935	0,1415
comp.graphics	0,100	150	114	239	0,5682	0,3856	0,4594
sci.electronics	0,000	257	1171	440	0,1800	0,3687	0,2419
soc.religion.christian	0,025	205	238	193	0,4628	0,5151	0,4875
sci.med	0,100	80	91	316	0,4678	0,2020	0,2822
sci.crypt	0,125	190	31	206	0,8597	0,4798	0,6159
alt.atheism	0,050	106	84	344	0,5579	0,2356	0,3313
misc.forsale	0,050	265	111	190	0,7048	0,5824	0,6378
comp.os.ms-windows.misc	0,100	195	133	199	0,5945	0,4949	0,5402

Tabulka 34 Výsledné hodnoty pro jednotlivé kategorie (NG-1000).

	Micro	Macro
Precision	0,4794	0,5337
Recall	0,3839	0,3768
F1	0,4264	0,4417

Tabulka 35 Celkové výsledky pro všechny kategorie (NG-1000).

8.3 Experimenty na kolekci Iris

Experimentování na této kolekci probíhalo podle stejných principů zmíněných výše. Na trénovací množině byly naučeny vektory pro různá thresholds. Na základě porovnání s validační množinou byly vybrány nejlepší hodnoty threshold pro kategorie (tabulka 36). Dále pomocí takto získaných hodnot znovu proběhlo naučení a následné otestování na testovací množině. V tabulce 37 jsou vidět výsledné hodnoty pro jednotlivé kategorie. Když se podrobně podíváme na výsledné hodnoty, tak jistě vyvodíme také celkové vyhodnocení je rovno jedné (u všech hodnocených kritérií).

threshold	Iris-setosa	Iris-versicolor	Iris-virginica
0,00	**1,0000**	0,7500	**0,9091**
0,10	1,0000	0,7500	0,9091
0,20	1,0000	0,7500	0,9091
0,30	1,0000	0,7500	0,9000
0,40	1,0000	0,7500	0,9000
0,50	1,0000	0,7500	0,9091
0,60	0,9524	0,8000	0,9000
0,70	1,0000	0,8000	0,9000
0,80	1,0000	0,8000	0,9000
0,90	1,0000	0,8182	0,9000
0,99	1,0000	**0,8235**	0,9000

Tabulka 36 Výběr threshold v kolekci Iris.

	threshold	TP	FP	FN	Precision	Recall	F1
Iris-setosa	0,00	10	0	0	1,0000	1,0000	1,0000
Iris-versicolor	0,99	10	0	0	1,0000	1,0000	1,0000
Iris-virginica	0,00	10	0	0	1,0000	1,0000	1,0000

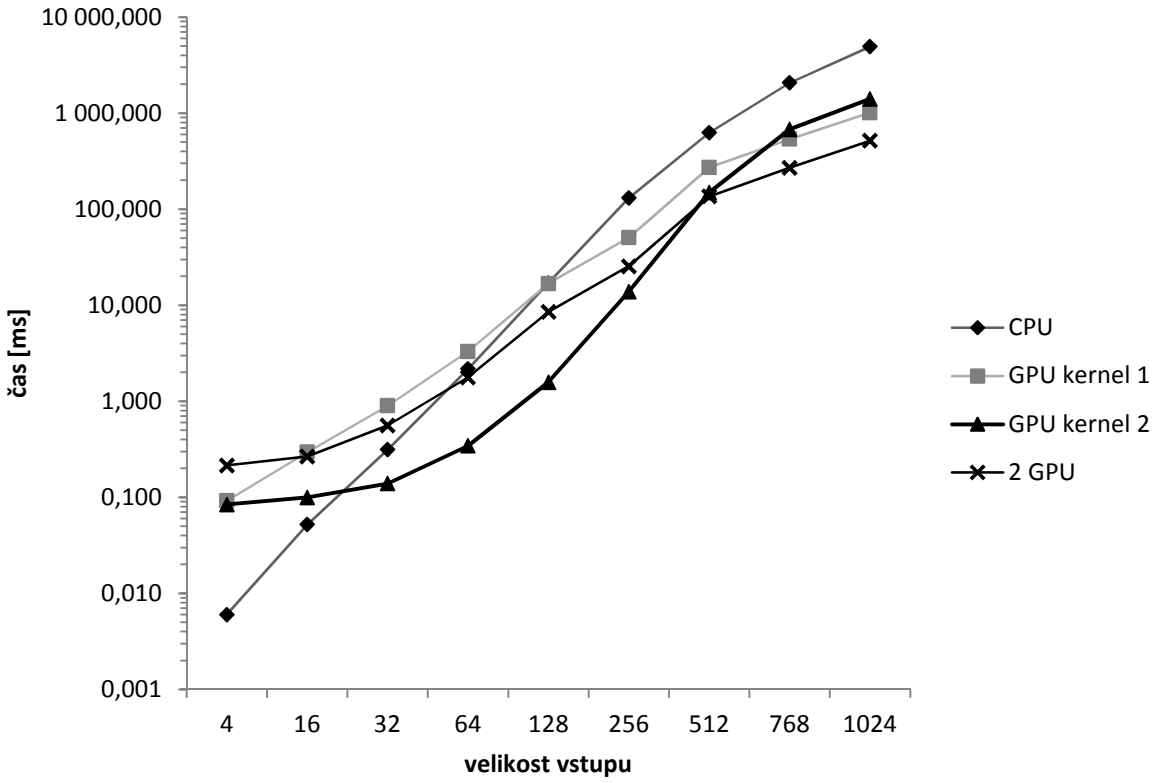
Tabulka 37 Výsledné hodnoty pro všechny kategorie (Iris).

8.4 Srovnání rychlosti implementací a výkonnostní testy

Jak ukazuje měření (v kapitole 6.2), největší podíl na časové náročnosti celého naučení vektorů pro dané kategorie má porovnávání m vektorů s n vektory. Proto se rozhodl provést výkonnostní testy pro tento kritický výpočet. Byly srovnány celkem čtyři funkce, které řeší tento problém. První je klasická implementace na CPU, druhá je implementace navržené metody kernel 1 (viz. 6.3), dále implementace metody označené jako kernel 2 (viz. 6.4) a nakonec implementace na dvou GPU, založená na kernel 1. Rychlost implementací se dá měřit po konfiguraci tří parametrů. Počet vektorů první množiny vektorů m, počet vektorů druhé množiny vektorů n a jejich dimenze d. V následujících obrázcích je za veli-

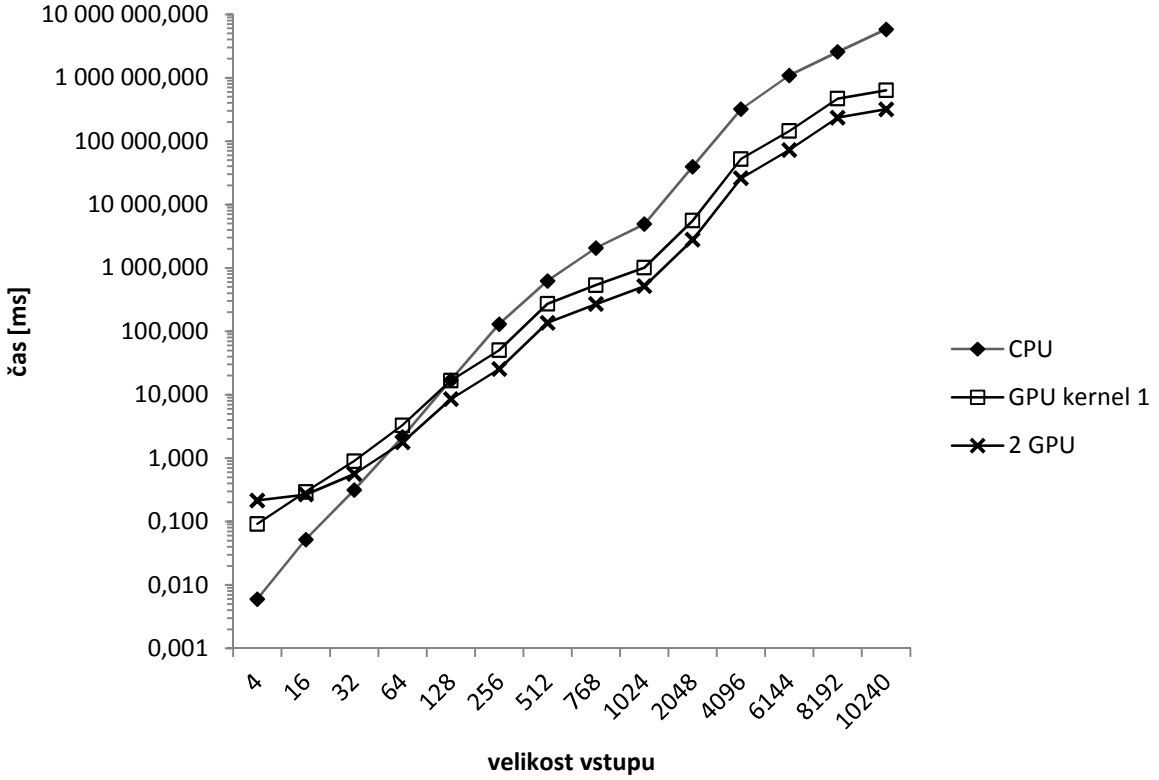
kost vstupu považované číslo, které je stejné pro m, n i d. Velikost vstupu 1024 znamená, že $d = m = n = 1024$. Aby šlo z grafu něco vyčíst, většinou byla pro časovou osu použita logaritmická míra.

V obrázku 8 je zachycena závislost velikosti vstupu na času. Z definice metody kernel 2 vyplývá omezení dimenze na hodnotu 1024 (více v kapitole 6.4). Proto je v grafu omezena maximální hodnota vstupu. Je zde přehledně vidět, že pro vstupy s velikostí od 32 do 256 je výhodnější použít GPU kernel 2. Pro menší vstupy CPU a pro větší vstupy implementaci 2 GPU.



Obrázek 8 Graf závislosti velikosti vstupu a času pro vstup do velikosti 1024.

Implementace GPU kernel 1 a 2 GPU je omezená pouze fyzickou pamětí dostupnou na GPU. V obrázku 9 jsou zobrazeny právě tyto dvě metody ve srovnání s CPU. Časová osa je zobrazena logaritmicky. Čas 2 GPU je zde oproti kernel 1 přibližně dvojnásobně rychlejší.



Obrázek 9 Graf závislosti vstupu a času pro větší vstupy.

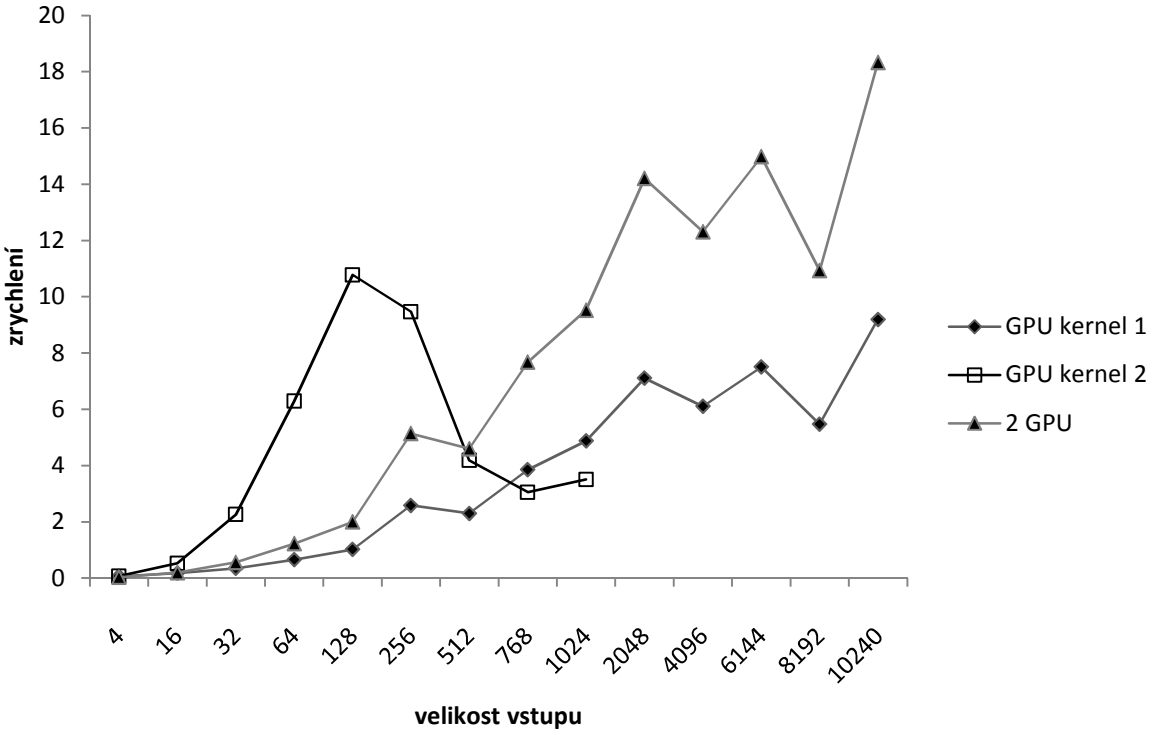
V tabulce 38 je přehledně vidět maximální dosažené zrychlení ve srovnání s implementací na CPU. Zrychlení do hodnoty jedna fakticky není zrychlení, ale zpomalení. Pro velice malé vstupy (4 až 32 prvků) je tak výhodnější použít klasický přístup pomocí CPU.

d=m=n	CPU	GPU kernel 1	GPU kernel 2	2 GPU	Max. zrychlení
4	0,01 ms	0,09 ms	0,08 ms	0,21	0,07
16	0,05 ms	0,30 ms	0,10 ms	0,27 ms	0,52
32	0,31 ms	0,90 ms	0,14 ms	0,56 ms	2,26
64	2,17 ms	3,30 ms	0,34 ms	1,77 ms	6,29
128	17,03 ms	16,76 ms	1,58 ms	8,52 ms	10,77
256	130,45 ms	50,52 ms	13,78 ms	25,40 ms	9,47
512	625,05 ms	271,64 ms	149,48 ms	135,92 ms	4,60
768	2,07 s	537,24 ms	678,77 ms	269,42 ms	7,68
1024	4,92 s	1,01 s	1,40 s	516,92 ms	9,53
2048	39,65 s	5,58 s		2,79 s	14,20
4096	320,16 s	52,42 s		26,00 s	12,31

6144	18,12 min	2,41 min		1,21 min	14,98
8192	42,74 min	7,82 min		3,91 min	10,93
10240	97,03 min	10,55 min		5,29 min	18,33

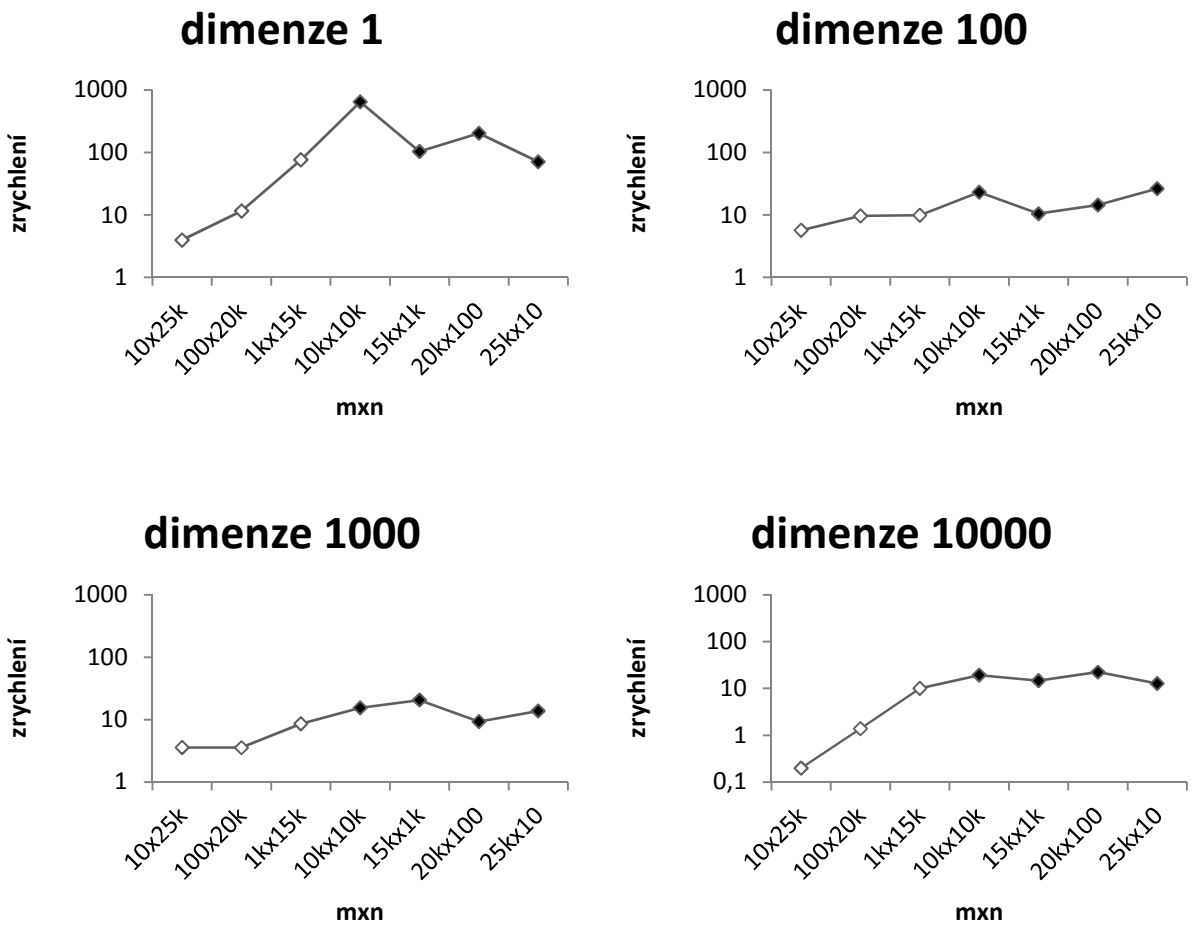
Tabulka 38 Časy a maximální zrychlení jednotlivých implementací.

V tabulce výše jsou vidět nevyplněné buňky, které plynou z definice GPU kernel 2. Stejně tak v grafu níže (obrázek 10) je zrychlení oproti GPU vyneseno pouze do omezující hodnoty 1024. V tomto grafu je přehledněji vidět zrychlení a výhodu použití GPU. Pro velikost vstupu 10240 je zrychlení více než osmnáctinásobné. Průměrné zrychlení je více než osminásobné.



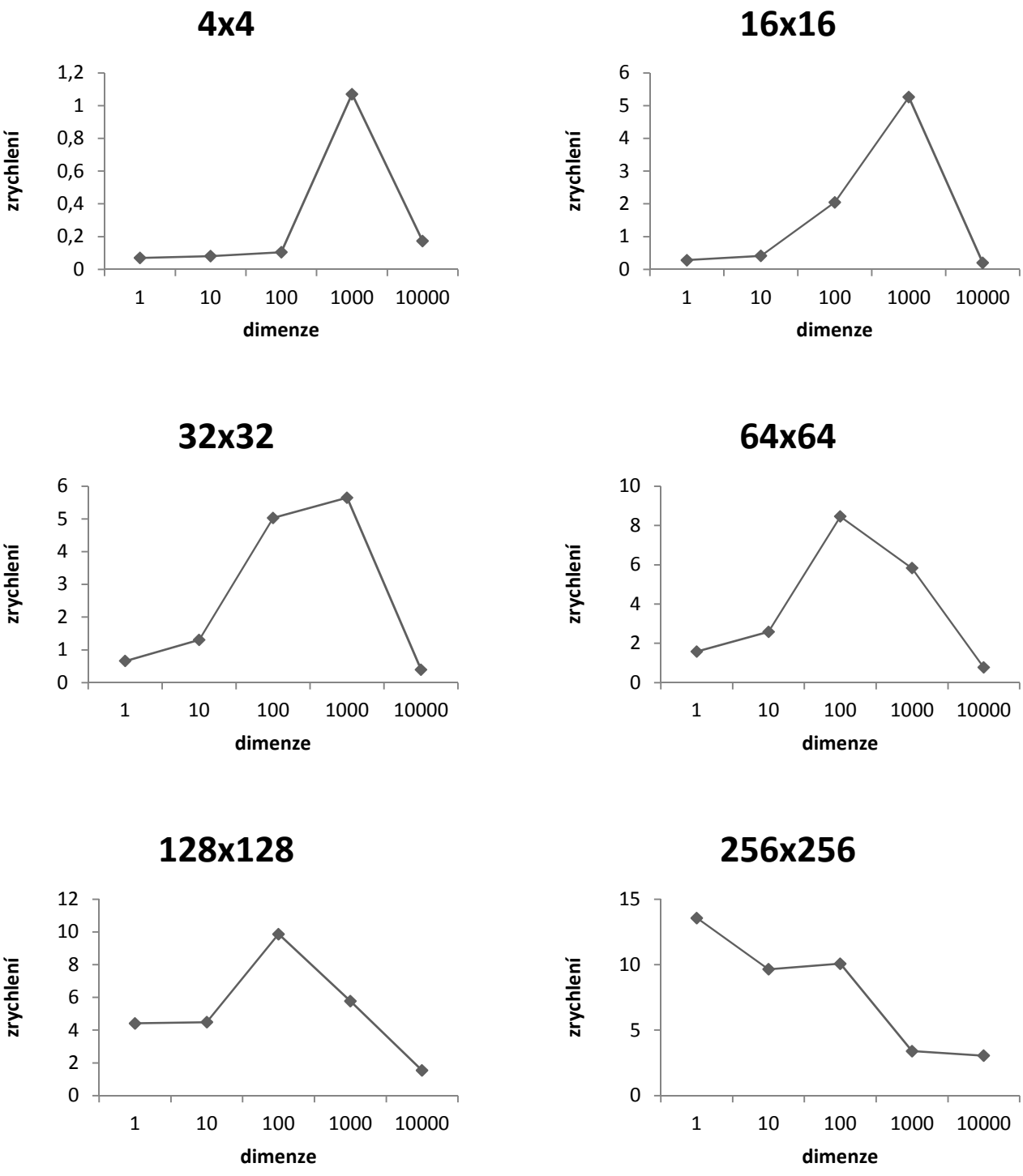
Obrázek 10 Maximální zrychlení v závislosti na velikosti vstupu.

Obrázek 11 se skládá ze čtyř grafů, ukazujících maximální zrychlení pro čtyři pevně stanovené dimenze a variabilní velikosti kolekcí. Implementaci kernel 1, respektive 2 GPU není jedno, v jakém pořadí jsou předány kolekce vektorů. Na vodorovné ose jsou různé velikosti kolekcí. V první polovině stoupá m a klesá n. Uprostřed jsou si velikosti rovny. Ve druhé polovině osy naopak klesá m a stoupá n. Zrychlení je zobrazeno vždy pro dvojici $a \times b$ i $b \times a$. Černě jsou vyplněná vždy maximální zrychlení pro každou z dvojic. Všechny grafy ukazují, že větší zrychlení nastává v případě, že $m > n$. Pokud se tedy do funkce v GPU pošlou kolekce ve správném pořadí, lze získat daleko větší zrychlení. Samozřejmě je nutné adekvátně změnit i konfiguraci kernel.

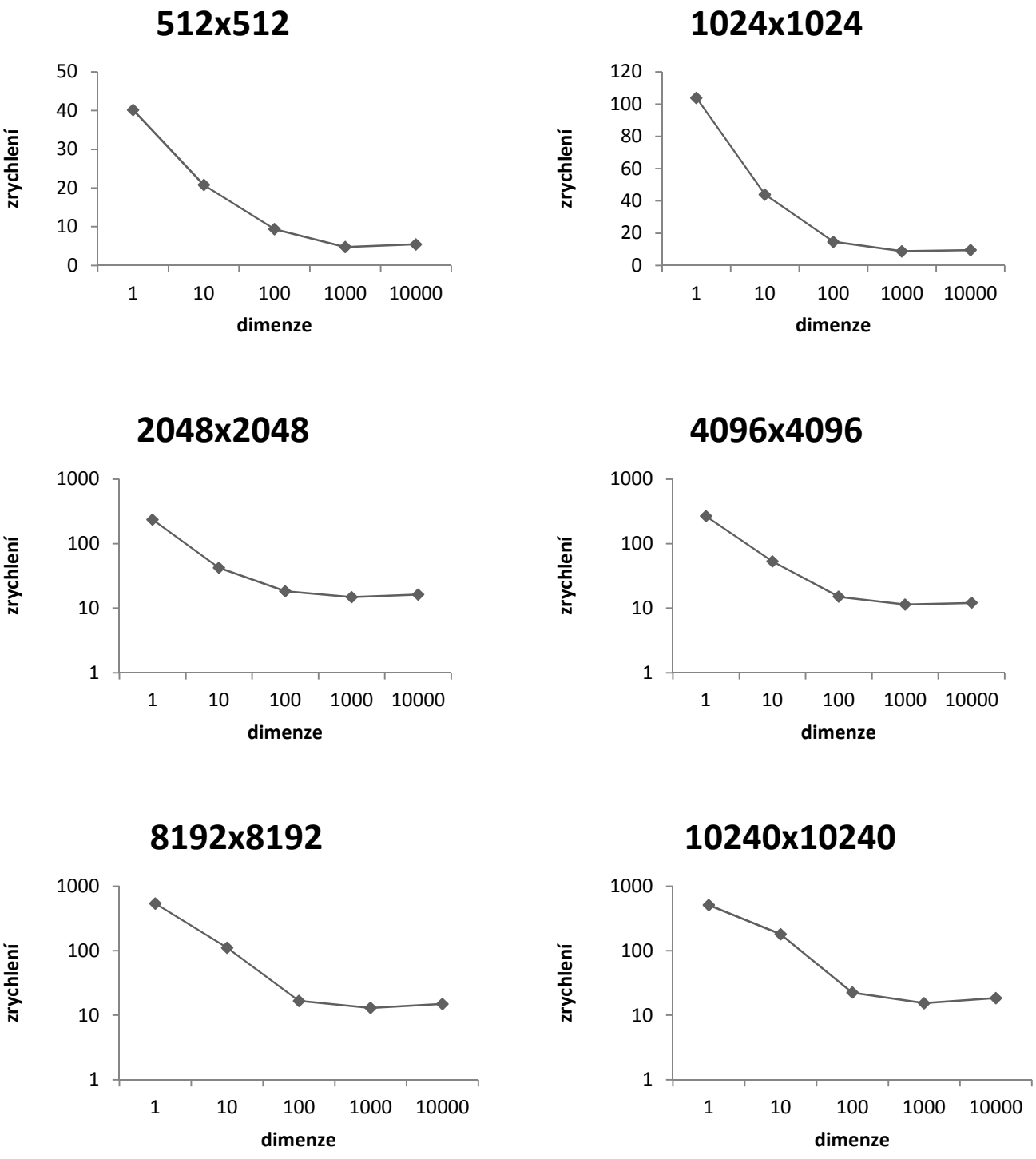


Obrázek 11 Závislost zrychlení na různě velkých vstupních vektorech se zafixovanou dimenzí.

V dalším měření byly zafixovány hodnoty m a n. V ose x se mění hodnota dimenze. Celé měření je zaznamenáno v grafech níže (obr.12 a 13). V nadpisu každého grafu je uveden počet porovnávaných vektorů z první množiny m a počet porovnávaných vektorů z druhé množiny o velikosti n. To, že velikosti rostou, jako násobky dvou není důležité. Samozřejmě stejně dobře umí vytvořený program zpracovat jakékoliv velikosti. Grafy potvrzují, že pro velmi malé vstupy (4x4, 16x16) GPU nějak nepomůže. Pro trochu větší vstupy nastává zrychlení, které ale pro největší testovanou hodnotu dimenze 10000 klesne. Od větších velikostí vstupů (128x128) a výše nastává vždy zrychlení. Největší zrychlení je pro data, ve kterých je dimenze co nejmenší. V implementaci kernelu 2 GPU vlákna provádějí cyklus napříč celým vektorem. Čím menší je dimenze vektoru a tedy samotný cyklus, tím méně náročné je to pro GPU.



Obrázek 12 Závislost dimenze na zrychlení (maximální) od 4x4 do 256x256.



Obrázek 13 Závislost dimenze na zrychlení (maximální) od 512x512 do 10240x10240.

8.5 Výkonnostní testy na kolekcích

V předešlé kapitole byla otestována nejvíce kritická část celého procesu. Nyní se ale podívejme, jak vypadá urychlení na konkrétních testovaných kolekcích. Byly otestovány kolekce Reuters (dimenze 300, 1000), kolekce Iris a kolekce Newsgroup (dimenze 300 a 1000). V tabulce 39 jsou vidět výsledné časy a zrychlení oproti CPU. Měření probíhalo vždy na kategorii a jedné iteraci. Počet jedinců byl vždy nastaven na hodnotu 10000. Pro naučení jedné kategorie bylo v předešlých experimentech provedeno vždy 15 takových iterací. U největší kolekce byl celkový čas výpočtu snížen z více než 32 hodin na necelé 4 hodiny. Bez použití GPU by bylo například ladění hodnoty threshold ještě více časově náročnější.

	m	n	d	CPU (ms)	GPU (ms)	zrychlení
r-ma-1000	4161	10000	1000	214 681,59	22 718,55	**9,45**
ng-300	7945	10000	300	136 503,43	16 729,22	**8,16**
ng-1000	7902	10000	1000	391 901,12	47 262,28	**8,29**
r-ma-300	4075	10000	300	79 501,35	7 607,99	**10,45**
iris	60	10000	4	112,80	44,35	**2,54**

Tabulka 39 Zrychlení na konkrétních kolekcích.

8.6 Použitý hardware a software

Pro veškeré testy bylo použito vybavení školního počítače Tesla01. Ten má k dispozici dvě GPU karty nVidia Tesla C2050 s celkem 896 jádry a 2x3 GB paměti. Dále má dva dvou-jádrové procesory AMD Opteron. Jako operační systém má počítač Linux CentOS. Program jsem primárně vyvíjel s použitím vývojového prostředí Microsoft Visual Studio, ale zajišťoval jsem také kompatibilitu s prostředím Linux (psaním „Makefile“).

9 Zhodnocení

Na kolekci Reuters byly provedeny čtyři větší experimenty. Experiment R-3fold-300 se celkem dobře dá srovnávat s experimentem provedeným v [4]. I když publikace neuvádí všechny parametry testu (například počet jedinců, počet generací, volba threshold atd.), lze provést alespoň přibližné srovnání. Srovnáním tabulky 25 a výsledků v [4] (tabulka 5) je vidět, že F_1 se liší o čtyři setiny. Algoritmus je totožný, stejných výsledků by se tedy dosáhlo změnou parametrů, které byly použity při experimentech. Manuální zkoušení různých parametrů by byl velice časově náročný úkol. Ve srovnání s jinými algoritmy pro klasifikaci uvedených v [4] se ale jedná stále o nejlepší výsledek. Experiment R-300 používá vlastní rozdělení dokumentů na trénovací, validační a testovací podmnožiny v poměru 60, 20 a 20. Tento experiment dosáhl z testovaných rozdělení nejlepších výsledků (tabulka 17). Dále experimenty označené jako R-ma-300 a R-ma-1000 testovaly standardní rozdělení kolekce Reuters. Při jejich porovnání je vidět rozdíl ve výsledcích při použití dimenze 300 a dimenze 1000 (tabulka 21 a 29).

Kolekce 20 Newsgroup je už méně zajímavá. Bylo zde otestováno chování na standardním rozdělení této kolekce pro dimenzi 300 a dimenzi 1000. Pro dimenzi 300 dosáhly výsledky hodnot 0,3549 respektive 0,3690 (tabulka 32), což je relativně málo. Pro dimenzi 1000 byly výsledky o poznání lepší. Tím se ukazuje nutnost použití většího počtu features pro tuto kolekci.

Poslední testovaná kolekce Iris dosáhla výsledku F_1=1,0, což je nejlepší možný výsledek vůbec. Kolekce ale obsahuje jen 150 záznamů a tak nelze brát výsledek příliš vážně. Při validační fázi měla účelová funkce F_1 pro „nejhorší“ kategorii Iris-versicolor hodnotu 0,8235. Testovací množina obsahovala jen 30 záznamů, takže zařazení všech rostlin do správných kategorií je pravděpodobnější, než kdyby testovací množina obsahovala například 3000 záznamů.

Nyní se podívejme na konkrétní dosažené zrychlení na testovacích kolekcích. Srovnání bylo provedeno na základě implementace na CPU a implementace na GPU a byl srovnáván celkový čas nutný pro naučení vektorů. Výsledné časy v tabulce 39 ukazují vždy několikanásobné urychlení GPU oproti klasické implementaci. Pro nejmenší kolekci Iris bylo zrychlení jen 2,54 násobné a tak je výhoda použití GPU diskutabilní. Pro ostatní kolekce dosahovalo zrychlení průměrně devítinásobku, což už je poměrně výrazné zrychlení, které dokáže ušetřit na kolekci Newsgroup i více než 10 dní testování (zkrácení z 12 dní na 36 hodin).

Výkonnostní testy odhalily ještě větší potenciál zrychlení pro kolekce s jiným charakterem dat. Měřením bylo zjištěno, že čím menší je dimenze vektoru, tím méně náročné je to pro GPU (tedy pro výše popsané algoritmy kernel 1, kernel 2). Přínos GPU tedy může být na jiných kolekcích ještě vyšší.

10 Závěr

Tato diplomová práce se zabývala problémem klasifikace textů pomocí stochastického evolučního algoritmu PSO a jeho urychlením pomocí grafických karet, které se zde používají pro negrafické výpočty.

První kapitoly popisují, proč je důležité předzpracovávání textů, definují problém klasifikace, pojmy jako trénovací a testovací množiny dokumentů a metody vyhodnocení klasifikátorů. Ve čtvrté kapitole je detailně popsán algoritmus PSO a jeho možné parametry. Prostředí CUDA je popsáno v kapitole 4. Následující kapitola popisuje současné metody klasifikace pomocí GPU.

Byl vytvořen program pro zpracování textů, který z textů odstraní méně důležité informace a převede dokumenty na vektory. Jiný program vytvořený v CUDA, přečte tyto kolekce. Pomocí konfiguračních souborů algoritmus PSO v několika iteracích najde nejvhodnější vektory takové, které rozeznávají jednotlivé kategorie. Hlavním cílem této práce bylo navrhnout a implementovat vlastní metodu ve snaze urychlit tento komplikovaný výpočet.

Celý proces byl nejprve implementován na straně CPU. Byla provedena analýza a měření dílčích částí výpočtu a jasně se ukázalo, kde leží nejkritičtější místo z hlediska časové náročnosti. V kapitole 6.2 bylo ukázáno, že případné zrychlení, kterého by se dosáhlo implementací jiných částí výpočtu, by se v celkovém čase projevilo minimálně. Byly navrženy dva základní přístupy k řešení tohoto problému. První metoda (označená kernel 1) vyniká ve velkých vstupních datech, metoda kernel 2 zase ukázala svou sílu při relativně malých vstupech. Byl také implementován přístup založený na kernel 1, který využívá možnosti současně použít více grafických karet. Všechny zmíněné implementace byly otestovány na různě velkých vstupních datech. Na velmi malých datech nebylo zaznamenáno žádné zrychlení. Maximální dosažené zrychlení v závislosti na charakteru dat přesáhlo hodnotu 500x. Na konkrétních testovaných kolekcích se pak zrychlení pohybuje od 2,54 do 10,45 násobku.

I přes urychlení, kterého bylo dosaženo, se ukázalo, že PSO nepatří k nejrychlejším klasifikátorům, alespoň co se týče fáze učení. Ve fázi testování naopak vyniká, protože dokument jednoduše zařadí, nebo nezařadí do kategorie pouhým porovnáním dvou vektorů a threshold.

Diplomová práce obsahuje řadu experimentů na třech testovaných kolekcích. Aby bylo ukázáno, že vytvořený program je univerzální a poradí si i s netextovými kolekcemi, byla otestována kolekce Iris, která zaznamenává údaje o rostlinách. Další standardní kolekce Reuters a 20 Newsgroup jsou textové. V kapitole 5 byl zpracován přehled současného stavu využití GPU pro klasifikaci dokumentů. V kapitole 9 byl zhodnocen přínos GPU.

Inspirací pro vznik této práce byla publikace „A PSO-Based Web Document Classification Algorithm", ve které byl tento algoritmus použit a otestován. Experimenty provedené v této práci potvrdily její výsledek. Algoritmus PSO dosáhl na testované kolekci lepších výsledků, než ostatní srovnávané algoritmy. Celý proces se však pomocí CUDA podařilo poměrně výrazně urychlit.

Vysoká škola báňská, speciálně Katedra informatiky je jedním z členů CUDA Research Center. Všechny testy byly provedeny na školním počítačovém vybavení, kde jsem měl k dispozici dvojici grafických karet podporujících prostředí CUDA.

Námětů na další práci v této oblasti by se dalo vymyslet hned několik. Velice zajímavé by bylo otestovat nějaký jiný optimalizační algoritmus na tomto problému. Například jakákoliv verze genetických algoritmů, nebo algoritmus diferenciální evoluce by problém dozajista také řešila. Z hlediska zrychlení výpočtu bych navrhoval vyzkoušet několik věcí. První by byla provedení výpočtu celého F_1 pomocí GPU, dále by se mohlo aplikovat urychlení PSO popsané v [16], které by také zrychlilo výpočet až o několik procent na větších datech. Pokud by bylo možné otestovat program na více než dvou GPU, dal by se také navrhnout sofistikovaněji způsob výpočtu na více GPU, který by přímo počítal s využitím více karet. Rychlost porovnávání vektorů byla pomocí GPU mnohonásobně zvýšena, a tak by při současné rychlosti HW také stálo za úvahu, použít celou trénovací množinu jako klasifikátor. Testovací data by se pak klasifikovala na základě všech trénovacích vektorů, které by měly určitý threshold. Další pokusy by mohly směrovat směrem k testování jiných přístupů k porovnávání vektorů (Euklidovská metrika, Manhattanská metrika, apod.). Na základě metriky by se samozřejmě daly optimalizovat i funkce na GPU.

Literatura

[1] Default English stopwords list. *Ranks NL*: *Webmaster Tools* [online]. [cit. 2012-04-26]. Dostupné z: http://www.ranks.nl/resources/stopwords.html

[2] PORTER, Martin. The Porter Stemming Algorithm. *Porter Stemming Algorithm* [online]. 2006 [cit. 2012-04-26]. Dostupné z: http://tartarus.org/~martin/PorterStemmer/

[3] *Compute Unified Device Architecture - Programming Guide 2.0*. [s.l.] : [s.n.], 2008. 107 s. Dostupný z WWW: <http://developer.download.nvidia.com/compute/cuda/2_0/docs/ NVIDIA_CUDA_Programming_Guide_2.0.pdf>.

[4] ZIQIANG, Wang, Zhang QINGZHOU a Zhang DEXIAN. A PSO-Based Web Document Classification Algorithm. *A PSO-Based Web Document Classification Algorithm* [online]. 2007, Eighth, s. 6 [cit. 2012-04-26]. DOI: 10.1109/SNPD.2007.72. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4287933

[5] SEBASTIANI, Fabrizio. A Tutorial on Automated Text Categorization. *A Tutorial on Automated Text Categorization* [online]. 1999, s. 25 [cit. 2012-04-26]. Dostupné z: http://web.iiit.ac.in/~jawahar/PRA-03/textCat.pdf

[6] ZELINKA, Ivan. *Evoluční výpočetní techniky*: *principy a aplikace*. 1. české vyd. Praha: BEN, 2009, 534 s. ISBN 978-80-7300-218-3.

[7] Reuters-21578 text categorization test collection: Distribution 1.0. *David D. Lewis - IR Consultant* [online]. 2004 [cit. 2012-04-26]. Dostupné z: http://www.daviddlewis.com/resources/testcollections/reuters21578/readme.txt

[8] SEBASTIANI, Fabrizio. Machine learning in automated text categorization. *ACM Computing Surveys* [online]. 2002, roč. 34, č. 1, s. 1-47 [cit. 2012-04-26]. ISSN 03600300. DOI: 10.1145/505282.505283. Dostupné z: http://portal.acm.org/citation.cfm?doid=505282.505283

[9] YANG, Yiming. An Evaluation of Statistical Approaches to Text Categorization. *Information Retrieval* [online]. 1999, roč. 1, 1/2, s. 69-90 [cit. 2012-04-26]. ISSN 13864564. DOI: 10.1023/A:1009982220290. Dostupné z: http://www.springerlink.com/openurl.asp?id=doi:10.1023/A:1009982220290

[10] SEBASTIANI, Fabrizio. Text Categorization. *Text Categorization* [online]. 2005, pp. 109-129, s. 21 [cit. 2012-04-26]. Dostupné z: http://nmis.isti.cnr.it/sebastiani/Publications/TM05.pdf

[12] SANDERS, Jason. *CUDA by example*: *an introduction to general-purpose GPU programming* [online]. 1st print. Upper Saddle River: Addison-Wesley, 2010, 290 s. [cit. 2012-04-26]. ISBN 978-0-13-138768-3. Dostupné z: http://www.amazon.com/CUDA-Example-Introduction-General-Purpose-Programming/dp/0131387685

[13] 20 Newsgroups. *MIT Computer Science and Artificial Intelligence Laboratory*: *CSAIL* [online]. 2008, 13:38:35 14.1.2008 [cit. 2012-04-26]. Dostupné z: http://people.csail.mit.edu/jrennie/20Newsgroups/

[14] FRANK, A. a A. ASUNCTION. Iris Data Set. CA: UNIVERSITY OF CALIFORNIA, School of Information and Computer Science. *UCI Machine Learning Repository* [online]. 2010 [cit. 2012-04-26]. Dostupné z: http://archive.ics.uci.edu/ml

[15] CUDA C Programming Guide Version: 4.2. *CUDA C Programming Guide Version* [online]. 2012, s. 173 [cit. 2012-04-26]. Dostupné z: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf

[16] GPU-based Parallel Particle Swarm Optimization. *GPU-based Parallel Particle Swarm Optimization* [online]. 2009, 1493 - 1500 [cit. 2012-04-26]. DOI: 10.1109/CEC.2009.4983119. Dostupné z: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4983119

[17] ZAHRAN, Ghassan. Text Feature Selection using Particle Swarm Optimization Algorithm. *Text Feature Selection using Particle Swarm Optimization Algorithm* [online]. [cit. 2012-04-26]. Dostupné z: http://www.idosi.org/wasj/wasj7%28c&it%29/10.pdf

[18] HARVEY, Jesse Patrick. GPU Acceleration of Object Classification Algorithms Using NVIDIA CUDA. GPU Acceleration of Object Classification Algorithms Using NVIDIA CUDA [online]. 2009 [cit. 2012-04-26]. Dostupné z: https://ritdml.rit.edu/bitstream/handle/1850/10894/35445_pdf_00B0B24A-DFD8-11DE-9A30-D21AD352ABB1.pdf?sequence=1

Adresářová struktura přiloženého DVD

/src	Zdrojové kódy vytvořených programů.
/src/Parser	Zdrojové kódy na předzpracování textů psané v jazyce Java.
/src/PSO-GPU	Program pro klasifikaci na GPU, testování a výkonnostní testování v CUDA.
/docs	Dokumentace k programům.
/data	Kolekce Reuters, 20 Newsgroup a Iris.
/results	Výsledky experimentů.
/text	Textová část diplomové práce v elektronické podobě.